# Slack Allocation Techniques for Intra-Path Load Balancing

Kartik Gopalan* Tzi-cker Chiueh Yow-Jian Lin
Binghamton University Stony Brook University Telcordia Technologies
Email: kartik@cs.binghamton.edu
Ph: 607-777-3751, Fax: 607-777-4729
Mail: Computer Science, Binghamton University, Binghamton, NY, USA, 13902–6000

***Abstract:*** Network resource provisioning techniques need to perform both inter-path and intra-path load balancing to maximize the network's resource usage efficiency while supporting end-to-end QoS guarantees. This paper focuses on the intra-path load balancing problem: How to partition the end-to-end QoS requirement of an aggregate network flow along the links of a given path such that the deviation in the loads on these links is as small as possible? We propose a set of new algorithms, called *Load-balancing Slack Allocation* (LSA), to solve this QoS partitioning problem for unicast and multicast traffic. The key concept in the LSA algorithms is the notion of *slack*, which quantifies the flexibility available in partitioning the end-to-end QoS requirement across the links of a selected path or tree. We show that one can improve network resource usage efficiency by carefully selecting a slack partition that explicitly balances the loads on the underlying links. These algorithms can simultaneously partition multiple QoS requirements such as delay and delay violation probability bounds. A detailed simulation study demonstrates that, compared with previous approaches, the LSA algorithms can increase the total number of long-term flows that can be provisioned along a network path by up to 1.2 times for deterministic and 2.8 times for statistical delay guarantees.

***Keywords:*** QoS Partitioning, Load-balancing, Resource Allocation.

*Corresponding author

# 1 Introduction

Emerging real-time network applications such as Voice over IP (VoIP), video conferencing, streaming media and online trading have stringent performance requirements in terms of end-to-end delay and throughput. A central problem faced by every network service provider is: *how to maximize the resource utilization efficiency of its physical network infrastructure and still guarantee the heterogeneous performance requirements of each customer*. Utilization efficiency is measured by the amount of customer traffic that can be supported over a fixed network infrastructure. Multi-Protocol Label Switched (MPLS) networks provide the flexibility to control resource allocation by explicitly selecting Label Switched Paths (LSPs) using signaling protocols such as RSVPTE [3] or LDP [1]. Each LSP can act as a traffic trunk that carries an aggregate traffic flow with Quality of Service (QoS) guarantees such as bandwidth and end-to-end delay bounds. We use the term *flow* to represent a *long-lived aggregate of multiple network connections* (such as a VoIP trunk) rather than short-lived individual streams (such as a single VoIP conversation).

The key approach underlying MPLS traffic engineering algorithms is to balance the loads on the network links and routers while allocating LSP paths [2, 18, 36]. Without load balancing, it is possible that resources at one link might be exhausted much earlier than others, thus rendering the entire network paths unusable. For aggregate real-time network flows that require end-to-end delay guarantees, there is an additional optimization dimension for load balancing besides path selection, namely, *the partitioning of end-to-end QoS requirements along the links of a selected network path*. The manner in which end-to-end QoS is partitioned among constituent links of network path directly impacts the extent of load imposed on each of the links. Specifically we are interested in the variable components of end-to-end delay, such as queuing delay at intermediate links and smoothing delay before the ingress, rather than the fixed delay components, such as propagation and switching delays. In this paper, we use the term "delay" to refer to *variable* components of end-to-end delay.

Consider the path shown in Figure 1 in which each link is serviced by a packetized rate-based scheduler such as WFQ [28]. Given a request for setting up a real-time flow $F$ along this path with a deterministic end-to-end delay requirement $D$, we need to assign delay budgets $D_l$ to $F$ at each individual link $l$ of the path. Intuitively, low delay typically requires high bandwidth reservation. In other words, since flow $F$'s packet delay bound at each link is inversely proportional to its bandwidth reservation, the amount of delay budget allocated to $F$ at a link determines the amount of bandwidth reservation that $F$ requires at that link.

The central question is, *how do we partition the end-to-end delay requirement $D$ into per-link delay budgets such that (a) the end-to-end delay requirement $D$ is satisfied and (b) the amount of traffic admitted along the multi-hop path can be maximized in the long-term?* This is the *Delay Budget Partitioning* problem for delay constrained network flows.

Most categories of real-time traffic, such as VoIP or video conferencing, can also tolerate their packets experiencing end-to-end delays in excess of $D$ within a small delay violation probability $P$. Such *statistical* delay requirements of the form $(D, P)$ can assist in reducing bandwidth reservation for real-time flows by exploiting their tolerance levels to delay violations. When we consider partition of the end-to-end delay violation probability require-

ment $P$, in addition to the delay requirement $D$, the delay budget partitioning problem is generalized to statistical delay requirements for network flows.

One of the important considerations in maximizing network resource utilization efficiency is to maintain load balance across different links. The prior approaches to solving the delay budget allocation problem were based on heuristics such as equal partitioning [25], proportional partitioning [8] or cost function based partitioning [6, 19, 21, 29] that did not directly consider the load balancing criteria. Additionally, the prior proposals handle partition of only a single QoS requirement and directly partition the entire end-to-end QoS over the links of the path. Our work makes the following main contributions.

- Firstly, we present a set of new algorithms, collectively called *Load-balancing Slack Allocation (LSA)*, to solve the delay budget partitioning problem, and describe its application to unicast and multicast flows with deterministic and statistical delay guarantees.

- Secondly, we introduce the notion of partitioning *slack* in end-to-end QoS rather than directly partitioning the entire end-to-end QoS as in earlier approaches. Slack quantifies the amount of flexibility available in balancing the loads across links of a multi-hop path and will be introduced in Section 2.

- Thirdly, our algorithm can handle multiple simultaneous flow QoS requirements such as end-to-end delay bounds and delay violation probability bounds rather than just a single end-to-end QoS requirement.

- Finally, we provide the detailed admission control algorithms for unicast and multicast flows that can be used in conjunction with any scheme of partitioning end-to-end delay and delay violation probability requirements.

We model our work on the lines of Guaranteed Service architecture [31] where the network links are serviced by packetized rate-based schedulers [28] [35] [9] [33] and there exists an inverse relationship between the amount of service rate of a flow at a link and the corresponding delay bound that the flow's packets experience. A rate-based model with GPS schedulers was also adopted in [8]. Note that we address the problem of partitioning *additive* end-to-end delay requirement and the *multiplicative* delay violation probability requirement. Indeed for packetized rate-based schedulers, such as WFQ, the end-to-end delay for a flow consists of both (additive) per-link packetization delays and (bottleneck) fluid queuing delay. In Section 4, we show how these two components can be separated and how the QoS partitioning problem is relevant in the context of rate-based schedulers.

Considering the bigger picture, any scheme for end-to-end QoS partitioning along a path is not sufficient by itself to satisfy traffic engineering constraints. Rather, end-to-end QoS partitioning is only one of the components of a larger network resource management system [10, 12] in which network resources need to be provisioned for each flow at three levels. At the first level, a network path is selected between a given pair of source and destination that satisfies the flow QoS requirements, and at same time balances the load on the network. At the second level, the end-to-end QoS requirement of the new flow is partitioned into QoS requirements at each of the links so as to balance the load along the selected path. This is the intra-path QoS partitioning problem that we address in this paper. Finally, at

the third level a resource allocation algorithm determines the mapping between the QoS requirements and the actual link-level resource reservation.

A conference version of this paper appeared earlier in [13]. In this paper we provide more detailed description and evaluation of our work, including the detailed partitioning algorithms for multicast paths and additional performance evaluations for general network topologies. The rest of the paper is organized as follows. Section 2 introduces the notion of *Slack Allocation* which is central to our work. Section 3 places our work in the context of related research in delay budget partitioning. Section 4 formulates our model of the network and reviews standard results for end-to-end delay bounds. In Section 5, 6 and 7 we describe a series of delay budget partitioning algorithms for unicast and multicast network paths having deterministic and statistical end-to-end delay requirements. In Section 8 we analyze the performance of the proposed algorithms and Section 9 concludes with a summary of main research results.

## 2 Notion of Slack and Slack Allocation

The extent of flexibility available in balancing the loads across links of a multi-hop path is quantified by the *slack* in end-to-end delay budget. For each link of the multi-hop path in Figure 1, we can compute the minimum local delay budget $D_l^{min}$ guaranteed to a new flow $F$ at the link $l$ provided that all residual (unreserved) bandwidth at $l$ is assigned for servicing packets from the new flow. The difference between the end-to-end delay requirement $D$ and the sum of minimum delay requirements $D_l^{min}$, i.e. $\Delta D = D - \sum_{l=1}^{3} D_l^{min}$, represents an excess *slack* that can be partitioned among the links to reduce their respective bandwidth loads.

The manner in which slack is partitioned among links of a flow path determines the extent of load balance (or imbalance) across the links. When the links on the selected network path carry different loads, one way to partition the slack is to allocate it based on the current loads on the network links. For example, assume that the three links in Figure 1 have resource utilization of $40\%$, $20\%$, and $80\%$ respectively. Given a total slack in delay budget $\Delta D$, one could partition the slack proportionally as $\frac{4}{14}\Delta D$, $\frac{2}{14}\Delta D$, and $\frac{8}{14}\Delta D$, respectively, rather than assigning each $\frac{1}{3}\Delta D$. The reason that the former assignment is better from the viewpoint of load-balancing is that a more loaded link should be assigned a larger delay budget in order to impose a lower bandwidth demand on it. The latter assignment, on the other hand, would lead to third link getting bottlenecked much earlier than the other two, preventing any new flows from being admitted along the path. In fact, as we will show later, we can do even better than proportional partitioning described above if we explicitly balance the loads across different links.

## 3 Related Work

In general, the QoS constraint partitioning problem is a special case of a general resource allocation problem that has been shown to be intractable [16, 29]. Thus all the solutions seek to find an approximate QoS constraint partition that satisfies certain optimization objectives. Previous solutions to this problem can be classified under two categories:

fixed partitioning and *cost-based* partitioning. We first summarize these approaches and then compare them against our contribution in this paper.

### 3.1 Fixed Partitioning

The fixed partitioning solutions, as the name suggests, apportion the end-to-end QoS among constituent links of a path using some form of fixed division criteria. Simplest form of fixed partitioning is the Equal Allocation (EA) approach [25], that divides the end-to-end loss rate requirement equally among constituent links of a flow path. The focus in [25] is on optimizing the minimal (bottleneck) link utility by equally partitioning the end-to-end loss rate requirements over the links. The performance of this scheme was shown to be reasonable for short paths with tight loss rate requirements but deteriorated for longer paths or higher loss rate tolerance. The principal conclusion in [25] is similar to ours, that is the key to maximize resource utilization is to reduce the load imbalance across network links. However, since EA does not take into account any information about link loads, it ends up treating a congested and a non-congested link with the same importance. Hence EA can potentially end up consuming excessive resources along bottleneck links.

The Proportional Allocation (PA) approach, proposed in [8], is a simple extension to the EA approach to QoS partitioning problem. It addresses the partitioning of end-to-end QoS over the constituent links of a multicast tree in proportion to the current utilization of each link. The performance of PA was shown to be better than EA since it accounts for differences in link loads.

In general, fixed partitioning approaches are also applicable in handling multiple simultaneous QoS requirements that are additive or multiplicative in nature. For instance, a real-time channel abstraction with deterministic and statistical delay bounds, based on a modified earliest deadline first (EDF) scheduling policy, has been proposed in [7]. This proposal uses the EA scheme to assign per-link delay and packet loss probability bounds.

### 3.2 Cost Function Based Solutions

Cost function based solutions attempt to search for a QoS-partition that minimizes or maximizes a global cost function, which is defined as the sum of local link costs. The cost function can be designed as a measure of some network optimization objective, such as load-balance across the links of a path. Efficient cost function based algorithms to partition the end-to-end QoS requirements of a unicast or multicast flow into per-link QoS requirements have been proposed in [6, 19, 21]. The cost functions are strongly convex in [19] and weakly convex in [21], whereas [6] addresses general cost functions. On the other hand, [29] addresses the problem in the context of discrete link costs in which each link offers only a discrete number of QoS guarantees and costs, such as in the framework of class-based discrete QoS in Diffserv. Work in [23] extends the results to the case of many-to-many multicast sessions. While the above approaches tackle the QoS partitioning problem in isolation, the combined problem of partitioning as well as QoS routing using cost function based approach has been addressed in [15, 22] where the goal is to maximize the probability of meeting the QoS requirements of a flow.

### 3.3 Comparison with LSA

Our proposal in this paper, called Load-balancing Slack Allocation (LSA), differs from the prior approaches in the following aspects. First, our proposal directly balances the loads on different links instead of indirectly addressing the objective via equal/proportional/cost function based partitioning. Secondly, the above proposals partition the entire end-to-end QoS over the links of the path. If a particular partition results in tighter delay requirement than the minimum possible at some link, then the proposal in [8] assigns the minimum delay at that link and then performs equal/proportional allocation over remaining links. With such an approach, the minimum delay assignment converts the corresponding link into a bottleneck, disabling all the paths that contain this link. In contrast, our LSA algorithm first identifies the *slack* in end-to-end QoS which quantifies the available flexibility in partitioning the end-to-end QoS. Instead of directly partitioning the end-to-end QoS, LSA proceeds to partition the slack which helps to prevent the formation of bottleneck links as long as non-zero slack is available.

For the cost function based algorithms to be effective, one needs to carefully devise a per-link cost function that accurately captures the global optimization objective – in our case that of maximizing number of flows admitted by balancing the loads among multiple links of the path. As we will demonstrate in Section 8, the best cost function that we could devise to capture the load balancing optimization criteria, when used with algorithms in [21], does not yield as high resource usage efficiency as the explicit load balancing approach proposed in this paper. Instead of indirectly addressing the load balancing objective via a cost function, our LSA algorithm explores only those partitions that maintain explicit load balance among links. Furthermore, the cost function based algorithms consider only single QoS dimension at a time. In contrast, our algorithm can handle multiple simultaneous QoS dimensions, such as both delay and delay violation probability requirements.

Earlier approach to address QoS partitioning as well as routing [15, 22] has different goal from ours, namely that of maximizing the probability of meeting the QoS requirements of a flow. While we focus only on the intra-path QoS partitioning problem in this paper, algorithms to integrate our proposal in this paper with QoS routing schemes has been described in [10, 12].

Algorithms were presented in [24] to select QoS-guaranteed end-to-end paths for traffic requiring bandwidth and delay guarantees. Approaches to provide end-to-end statistical performance guarantees have also been proposed in [4, 34] when rate controlled service discipline is employed inside the network. However, the above works do not specifically address the issue of how to locally partition the end-to-end delay requirement in order to maximize the number of admitted flows.

Another important context in which the delay budget partitioning problem arises is that of co-ordinated multi-resource allocation in real-time operating systems [11]. Here each system component, such as CPU, disk, or network interface, constitutes a resource instance, analogous to the earlier example of multiple network links along a network flow path. Real-time applications need end-to-end delay guarantees across the usage of multiple system resources. The goal of resource allocation is to assign delay budgets to tasks using different system resources such that the number of real-time applications admitted in the long term can be maximized.

# 4 Network Model

A *real-time flow* $F$ is defined as an aggregate that carries traffic with an average bandwidth of $\rho^{avg}$ and burst size $\sigma$. We assume that the amount of flow $F$ traffic arriving into the network in any time interval of length $\tau$ is bounded by $(\sigma + \rho^{avg}\tau)$. The $(\sigma, \rho^{avg})$ characterization can be achieved by regulating $F$'s traffic with relatively simple leaky buckets. We focus this discussion on unicast flows and will generalize to multicast flows when necessary.

We consider the framework of smoothing at the network ingress and bufferless multiplexing in the network interior as advocated in [30] [9]. Specifically, as shown in Figure 2, a regulated flow $F$ first traverses a traffic smoother followed by a set of rate-based link schedulers at each of the intermediate links along its multi-hop path. The first component smooths the burstiness in $F$'s traffic before the ingress node. Each rate-based scheduler at intermediate link $l$ services the flow at an assigned rate $\rho_l \geq \rho^{avg}$. The manner in which rates $\rho_l$ are assigned will be described later in Sections 5 and 6. The smoother regulates flow $F$'s traffic at a rate $\rho^{min} = \min_l\{\rho_l\}$ i.e., at the smallest of per-link assigned rates. Since flow $F$'s service rate at each link scheduler is greater or equal to smoothing rate at the ingress, $F$'s traffic does not become bursty at any of the intermediate links. Completely smoothing $F$'s traffic before the ingress node has the advantage that it allows the interior link multiplexers to employ small buffers for packetized traffic. Additionally, as shown below, it permits decomposition of flow's end-to-end delay requirement $D$ into delay requirements at each network component along the flow path. Multicast flows have a similar setup except that flow path is a tree in which each node replicates traffic along outgoing branches to children.

We now proceed to identify different components of end-to-end delay experienced by a flow $F$. The first component is the *smoothing delay*. The worst-case delay experienced at the smoother by a packet from flow $F$ can be shown to be as follows [9].

$$D_s = \sigma/\rho^{min} \tag{1}$$

The maximum input burst size is $\sigma$, the output burst size of the smoother is $0$, and the output rate of the smoother is $\rho^{min} = \min_l\{\rho_l\}$

The second component of end-to-end delay is the accumulated *queuing delay* at intermediate links. We assume that packets are serviced at each link by the Weighted Fair Queuing (WFQ) [5] [28] scheduler which is a popular approximation of the Generalized Processor Sharing (GPS) [28] class of rate-based schedulers. It can be shown [28] that the worst-case queuing delay $D_l$ experienced at link $l$ by any packet belonging to flow $F$ under WFQ service discipline is given by the following.

$$D_l = \frac{\delta_l}{\rho_l} + \frac{L_{max}}{\rho_l} + \frac{L_{max}}{C_l} \tag{2}$$

$\delta_l$ is $F$'s input burst size at link $l$, $L_{max}$ is the maximum packet size, $\rho_l$ is the reservation for $F$ at link $l$, and $C_l$ is the total capacity of link $l$. The first component of the queuing delay is fluid fair queuing delay, the second component is the packetization delay, and the third component is scheduler's non-preemption delay. Since our network model employs bufferless multiplexing at interior links, the input burst $\delta_l$ is $0$ at each link $l$. The delay bound of Equation 2 also holds in the case of other rate-based schedulers such as Virtual Clock [35]. In general, for any rate-based scheduler, we assume that a function of the form $\mathcal{D}_l(.)$ exists that correlates the bandwidth reservation $\rho_l$ on a link

$l$ to its packet delay bound $D_l$, i.e. $D_l = \mathcal{D}_l(\rho_l)$. We are interested in rate-based schedulers since, in their case, the relationship between per-link delay bound and the amount of bandwidth reserved at the link for a flow can be explicitly specified. In contrast, even though non rate-based schedulers (such as Earliest Deadline First (EDF) [20]) can potentially provide higher link utilization, in their case the resource-delay relationship for each flow is difficult to determine, which in turn further complicates the admission control process.

In Figure 2, the end-to-end delay bound $D$ for flow $F$ over an $m$-hop path is given by the following expression [26] [27] when each link is served by a WFQ scheduler.

$$D = \frac{\sigma}{\rho^{min}} + \sum_{l=1}^{m} \left( \frac{L_{max}}{\rho_l} + \frac{L_{max}}{C_l} \right) \tag{3}$$

Here $\rho_l \geq \rho^{avg}$ and $\rho^{min} = \min_l\{\rho_l\}$. In other words, end-to-end delay is the sum of traffic smoothing delay and per-link queuing (packetization and non pre-emption) delays. For multicast paths, end-to-end delay is the sum of smoothing delay at ingress and the maximum end-to-end queuing delay among all unicast paths from the source to the leaves.

# 5 Unicast Flow with Deterministic Delay Guarantee

We now propose a series of algorithms for delay budget partitioning that we call Load-balancing Slack Allocation (LSA) algorithms. The first algorithm, presented in this section, addresses deterministic delay guarantees for unicast flows. The second algorithm addresses statistical delay guarantees for unicast flows and the third algorithm extends LSA for multicast flows; these are presented in subsequent sections. The deterministic and statistical algorithms for unicast flows are named D-LSA and S-LSA respectively and those for multicast flows are named D-MLSA and S-MLSA.

Let us start with the case where flow $F$ is requested on a unicast path and requires an end-to-end deterministic delay guarantee of $D_F$ i.e, none of the packets carried by $F$ can exceed the delay bound of $D_F$. Assume that the network path chosen for a flow request $F$ consists of $m$ links and that $N-1$ flows have already been admitted on the unicast path. The total capacity and current utilized bandwidth on the $l^{th}$ link are represented by $C_l$ and $U_l$ respectively.

The goal of delay budget partitioning is to apportion $F$'s end-to-end delay budget $D_F$ into a set of delay budgets $D_{F,l}$ on the $m$ network links and the smoothing delay $D_{F,s}$ at the smoother, such that the following partition constraint is satisfied

$$D_{F,s} + \sum_{l=1}^{m} D_{F,l} \leq D_F \tag{4}$$

and *the number of flows that can be admitted over the unicast path in the long-term is maximized.*

We saw in Section 4 that for rate-based schedulers like WFQ, there exists a function of the form $\mathcal{D}_l(\rho_{F,l})$ that correlates a flow $F$'s bandwidth reservation $\rho_{F,l}$ on a link $l$ to its packet delay bound $D_{F,l}$. The specific form of relation $\mathcal{D}_l(\rho_{F,l})$ is dependent on the packet scheduling discipline employed at the links of the network. For example, if a link is managed by a WFQ scheduler, then the relation is given by Equation 2.

### 5.1 Admission Control (D-ADM)

Before computing $D_{F,l}$, one needs to determine whether the flow $F$ can be admitted into the system in the first place. Towards this end, first we calculate the minimum delay budget that can be guaranteed to $F$ at each link if all the residual bandwidth on the link is assigned to $F$. Thus the minimum delay budget at link $l$ is given by $\mathcal{D}_l(C_l - U_l)$, where $C_l$ is the total capacity and $U_l$ is the currently reserved capacity. From Equation 1, the corresponding minimum smoothing delay is $\sigma_F / \min_l\{C_l - U_l\}$. The flow $F$ can be admitted if the sum of minimum smoothing delay and per-link minimum delay budgets is smaller than $D_F$; otherwise $F$ is rejected. More formally,

$$\frac{\sigma_F}{\min_l\{C_l - U_l\}} + \sum_{l=1}^{m} \mathcal{D}_l(C_l - U_l) \leq D_F \tag{5}$$

**Algorithm 1** D-LSA: Load-balancing Slack Allocation algorithm for a unicast flow with deterministic delay guarantee. The algorithm returns the delay vector $\hat{D_F} =< D_{F,1}, D_{F,2}, \cdots, D_{F,m} >$.

**Algorithm D-LSA($F$, $D_F$)**
/* $F$ = Unicast flow being admitted. */
/* $D_F$ = End-to-end delay bound. */

$\delta = \frac{1}{2} \min_l \frac{C_l - U_l}{w_l C_l}$;
$b = \delta$;
**while** (1) **do**
  **for** $l = 1$ to $m$ **do**
    $\rho_{F,l} = max\{(C_l - U_l - w_l C_l b),\ \rho_F^{avg}\}$
    $D_{F,l} = \mathcal{D}_l(\rho_{F,l})$; /* Delay at link $l$ */
  **end for**

  $D_{F,s} = \sigma_F / \min_l\{\rho_{F,l}\}$; /* Smoother delay */

  $slack = D_F - \sum_{l=1}^{m} D_{F,l} - D_{F,s}$;

  **if** ($slack \geq 0$ and $\delta \leq \epsilon$) **then**
    return $\hat{D_F}$;
  **end if**

  $\delta = \delta/2$;

  **if** ($slack > 0$) **then**
    $b = b + \delta$;
  **else**
    $b = b - \delta$;
  **end if**
**end while**

### 5.2 Load-balancing Slack Allocation (D-LSA)

Once the flow $F$ is admitted, next step is to determine its actual delay assignment at each link along the unicast path. We define the *slack* in delay as

$$\Delta D_F = D_F - D_{F,s} - \sum_{l=1}^{m} D_{F,l} \tag{6}$$

If the flow $F$ can be admitted, it means that after assigning minimum delay budgets to the flow at each link, the slack $\Delta D_F$ is positive. The purpose of slack allocation algorithm (D-LSA) is to reduce the bandwidth requirement of a new flow $F$ at each of the $m$ links in such a manner that the number of flows admitted in future can be maximized. A good heuristic to maximize number of flows admissible in future is to apportion the slack in delay budget across multiple links traversed by the flow such that the load across each intermediate link remains balanced. By minimizing the load variation, the number of flow requests supported on the network path can be maximized.

The D-LSA algorithm for unicast flows with deterministic delay guarantee is given in Algorithm 1. Let the remaining bandwidth on the $l^{th}$ link after delay budget assignment be the form $w_l \times C_l \times b$, where $w_l \geq 1$ is a weighting factor for link $l$ and $b$ is a load balancing knob that determines the partitioning of slack ($0 \leq b \leq \min_l \frac{C_l - U_l}{w_l C_l}$). The algorithm essentially tunes the value of $b$ in successive iterations until the $b$ falls below a predefined threshold $\epsilon$; smaller the value of $\epsilon$, the closer LSA can get to the optimization objective.

Note that Algorithm 1 is essentially a *bisection method* [32] that finds the location of the root (i.e., $slack = 0$) by repeatedly dividing an interval in half and then selecting the sub-interval in which the root exists. This algorithm has $O(m \log \frac{1}{\epsilon})$ complexity [32] because each iteration has a complexity of $O(m)$ and the number of iterations is given by $\log(k/\epsilon)$, where the constant $k = \min_l \frac{C_l - U_l}{w_l C_l}$ is the maximum length of the search interval. In order for the bisection method to terminate, the corresponding $slack$ computation function needs to be monotonic with respect to the load-balancing knob $b$. This can be easily verified as follows. Line 14 in Algorithm 1 can be expanded as $slack = D_F - \sum_1^m (L_{max}/\rho_{F,l} + L_{max}/C_l) - \sigma_F / \min_l\{\rho_{F,l}\}$. Thus, $slack$ is monotonic w.r.t. $\rho_{F,l}\ \forall\ l$. From line 9 in Algorithm 1, we can see that $\rho_{F,l}$ is monotonic w.r.t. $b$, and hence $slack$ is monotonic w.r.t. $b$. Consequently, the bisection method is guaranteed to terminate.

Since $w_l \times C_l \times b$ represents the remaining capacity of the $l^{th}$ link, $w_l$ can be set differently depending on the optimization objective. If the optimization objective is to ensure that a link's remaining capacity is proportional to its raw link capacity, $w_l$ should be set to 1. This indeed turns out to be the best strategy when we are allocating resources on a unicast path in isolation, since it leads to perfect load balancing among the links. On the other hand, this strategy may not be the best approach for a general network topology where links happen to be shared among intersecting paths. Thus the value of $w_l$ needs to be carefully chosen depending upon the *network-wide* optimization objective rather than one local to the unicast path. For the objective of maximizing the amount of admitted traffic in the network, we need to ensure that resources at links, that are expected to shoulder more traffic load than other links, should last longer. We will show in Section 8 that one effective strategy in a general network topology is to select $w_l = 1/u_l$, where $0 \leq u_l \leq 1$ is the long-term utilization for link $l$.

# 6 Unicast Flow with Statistical Delay Guarantee

Now we consider the case where a new flow $F$ requires statistical delay guarantees $(D_F, P_F)$ over a $m$-hop unicast path, i.e, the end-to-end delay of its packets needs to be smaller than $D_F$ with a probability greater than $1 - P_F$. Since the delay bound does not need to be strictly enforced at all times, the network resource demand can be presumably

smaller for a fixed $D_F$. The S-LSA algorithm needs to distribute $D_F$ and $P_F$ to constituent links of the $m$-hop path. In other words, it needs to assign values $D_{F,l}$ and $P_{F,l}$, such that the following partition constraints are satisfied

$$D_{F,s} + \sum_{l=1}^{m} D_{F,l} \leq D_F \tag{7}$$

$$\prod_{l=1}^{m} (1 - P_{F,l}) \geq (1 - P_F) \tag{8}$$

and *the number of flow requests that can be admitted into the system in the long-term is maximized.* Here we assume there exist correlation functions $\mathcal{D}_l(\rho_{F,l}, P_{F,l})$ and $\mathcal{P}_l(\rho_{F,l}, D_{F,l})$ that can correlate the bandwidth reservation $\rho_{F,l}$ to a statistical delay bound $(D_{F,l}, P_{F,l})$.

$$D_{F,l} = \mathcal{D}_l(\rho_{F,l}, P_{F,l}) \tag{9}$$

$$P_{F,l} = \mathcal{P}_l(\rho_{F,l}, D_{F,l}) \tag{10}$$

In Section 4, we gave a concrete example of such correlation functions in the context of deterministic delay guarantees where link was serviced by a WFQ scheduler. At the end of this section, we will provide an example of how such correlation functions can be determined for statistical delay guarantees using measurement based techniques.

Note that the above condition on partitioning the end-to-end delay violation probability is more conservative than necessary. In particular, we assume that a packet can satisfy its end-to-end delay bound only if it satisfies its per-hop delay bounds. For instance a packet could exceed its delay bound at one link, be serviced early at another link along the path and in the process still meet its end-to-end delay bound. However, modeling such a general scenario is difficult and depends on the level of congestion at different links and their inter-dependence. Hence we make the most conservative assumption that a packet which misses its local delay bound at any link is dropped immediately and not allowed to reach its destination. This helps us partition the end-to-end delay violation probability into per-hop delay violation probabilities as mentioned above.

### 6.1 Admission Control (S-ADM)

The admission control algorithm for the case of statistical delay guarantees is more complicated than the deterministic case because it needs to check whether there exists at least one set of $\{< D_{F,l}, P_{F,l} >\}$ that satisfy the partitioning constraints 7 and 8. The detailed admission control algorithm is shown in Algorithm 2. It starts with an initial assignment of minimum delay value $\mathcal{D}_l(C_l - U_l, 0)$ to $D_{F,l}$ assuming that all the remaining capacity of the $l^{th}$ link is dedicated to $F$ and $P_{F,l} = 0$. Since the initial assignment might violate end-to-end delay constraint, i.e. $D_{F,s} + \sum_{l=1}^{m} D_{F,l} > D_F$, the algorithm attempts to determine if there exists a looser $P_{F,l}$ such that the delay partition constraint can be satisfied. Thus the algorithm iteratively increases the value of some $P_{F,k}$ by a certain amount $\delta$ and recomputes its associated $D_{F,k}$, until either $D_{F,s} + \sum_{l=1}^{m} D_{F,l}$ becomes smaller than $D_F$, or $\prod_{l=1}^{m} (1 - P_{F,l})$ becomes smaller than $(1 - P_F)$. In the first case, $F$ is admitted since a constraint satisfying partition exists. In the latter case even assigning all the available resources along the $F$'s path is insufficient to support the QoS requested

**Algorithm 2** S-ADM: The admission control algorithm for unicast flow $F$ with statistical delay requirements $(D_F, P_F)$.

**Algorithm S-ADM($F$, $D_F$, $P_F$)**
/* $F$ = Unicast flow being admitted. */
/* $D_F$ = End-to-end delay bound. */
/* $P_F$ = End-to-end delay violation probability bound. */

**for** $l = 1$ to $m$ **do**
  $P_{F,l} = 0$;
  $D_{F,l} = \mathcal{D}_l(C_l - U_l, 0)$;
**end for**
**while** ( $(D_{F,s} + \sum_{l=1}^{m} D_{F,l} > D_F)$ and $(\prod_{l=1}^{m}(1 - P_{F,1}) \geq (1 - P_F))$ ) **do**
  $k$ = index of link such that reduction in delay $D_{F,k} - \mathcal{D}_k(C_k - U_k, P_{F,k} + \delta)$ is maximum among all links;
  $P_{F,k} = P_{F,k} + \delta$;
  $D_{F,k} = \mathcal{D}_k(C_k - U_k, P_{F,k})$;
**end while**
**if** $(\prod_{l=1}^{m}(1 - P_{F,l}) < (1 - P_F))$ **then**
  Reject flow request $F$;
**else**
  Accept flow request $F$;
**end if**

and $F$ is rejected. In each iteration, that link $k$ is chosen whose delay budget reduces the most when its violation probability bound is increased by a fixed amount, $\delta$, i.e, the one that maximizes $D_{F,k} - \mathcal{D}_k(C_k - U_k, P_{F,k} + \delta)$.

The complexity of this admission control algorithm is $O(mn)$, where $m$ is the number of links and $n$ is the number of iterations in the $while$ loop, the latter being dependent upon how soon either the end-to-end delay falls below $D_F$ or the delay satisfaction probability falls below $(1 - P_F)$. Since each iteration increments the value of some $P_{F,k}$ by a positive $\delta$, the loop is guaranteed to terminate.

### 6.2 Load-balancing Slack Allocation (S-LSA)

In the context of statistical delay guarantees, the goal of slack allocation algorithm is to apportion both the slack in delay $\Delta D_F$ and the slack in assigned probability $\Delta P_F$ over $m$ network links traversed by the flow $F$. $\Delta D_F$ is calculated using Equation 6 and $\Delta P_F$ is calculated as follows.

$$\Delta P_F = \frac{\prod_{l=1}^{m}(1 - P_{F,l})}{(1 - P_F)} \tag{11}$$

Let the delay vector $< D_{F,1}, D_{F,2}, \cdots, D_{F,m} >$ be represented by $\hat{D_F}$. Similarly, let $\hat{P_F}$ represent the probability vector $< P_{F,1}, P_{F,2}, \cdots, P_{F,m} >$. The S-LSA heuristic algorithm for unicast flows with statistical delay guarantees is given in Algorithm 3. The algorithm starts with a feasible assignment of minimum delay vector $\hat{D_F}$ and probability vector $\hat{P_F}$ obtained during admission control. In every iteration, the algorithm first relaxes the delay vector $\hat{D_F}$ assuming fixed probability vector $\hat{P_F}$, and then relaxes the probability vector while fixing the delay vector. This process repeats itself till either the vector distance between the two values of $\hat{D_F}$ and between two values of $\hat{P_F}$ across two consecutive iterations falls below a predefined threshold, or the solution does not converge within

$max\_iterations$. In this heuristic, since the $\hat{P_F}$ values affect the $\hat{D_F}$ value relaxation step and vice-versa, multiple rounds of alternating relaxation steps are typically required to arrive at the final partition. The complexity of this algorithm is $O(nm\log(1/\epsilon))$, where $n$ is the number of iterations, $m$ is the number of links, and $\epsilon$ is the cutoff threshold within the two bisection algorithms. In evaluations described in Section 8, we empirically observe that this two-step iterative relaxation algorithm typically converges to a solution within 2 to 5 iterations.

The `relax_delay()` procedure is similar to the deterministic D-LSA algorithm in Algorithm 1 except that the resource correlation function $\mathcal{D}_l(\rho_{F,l})$ is replaced by $\mathcal{D}_l(\rho_{F,l}, P_{F,l})$. Algorithm 4 gives the `relax_prob()` procedure which is similar to `relax_delay()`, except that the correlation function is $\mathcal{P}_l(\rho_{F,l}, D_{F,l})$ and the slack in probability is defined as in Equation 11.

As in the case of Algorithm 1, the bisection method in Algorithm 4 has $O(m\log\frac{1}{\epsilon})$ complexity [32]. Furthermore, the algorithm will terminate only if the corresponding $slack$ computation function is monotonic w.r.t. the load balancing knob $b$. Following the logic similar to Section 5.2, line 13 in Algorithm 4 shows that $slack$ is monotonic w.r.t $P_{F,l}\ \forall\ l$. Also, we show in Equation 13 below that $P_{F,l} = \mathcal{P}_l(\rho_{F,l}, D_{F,l})$ is monotonic w.r.t. $\rho_{F,l}$. From line 10 in Algorithm 4, we can see that $\rho_{F,l}$ is monotonic w.r.t. $b$. Thus by inference, $slack$ in Algorithm 4 is monotonic w.r.t. $b$ and the bisection method is guaranteed to terminate.

**Algorithm 3** S-LSA: Load-balancing Slack Allocation algorithm for unicast flows with statistical delay guarantee.

1: **Algorithm S-LSA($F$, $D_F$, $P_F$)**
2: /* $F$ = Unicast flow being admitted. */
3: /* $D_F$ = End-to-end delay bound. */
4: /* $P_F$ = End-to-end delay violation probability bound. */
5:
6: Initialize $\hat{D_F}$ and $\hat{P_F}$ with the final $D_{F,l}$ and $P_{F,l}$ values computed from the admission control algorithm;
7: $iterations = 0$;
8: **do**
9: $\quad \hat{D_F'} = \hat{D_F};\ \hat{P_F'} = \hat{P_F}$;
10: $\quad \hat{D_F}$ =relax_delay($F$, $\hat{P_F}$, $D_F$);
11: $\quad \hat{P_F}$ =relax_prob($F$, $\hat{D_F}$, $P_F$);
12: $\quad iterations = iterations + 1$;
13: **while**( ((| $\hat{D_F} - \hat{D_F'}$ |$> thresh_D$) OR (| $\hat{P_F} - \hat{P_F'}$ |$> thresh_P$)) AND ($iterations\ <\ max\_iterations$));

## 6.3 Delay to Resource Correlation

In this section, we briefly give an example of a link-level mechanism to determine the correlation functions $\mathcal{D}_l(.)$ and $\mathcal{P}_l(.)$ for statistical delay guarantees using measurement based techniques. The approach, called Delay Distribution Measurement (DDM) based admission control, is described in detail in [14]. The DDM approach reduces the resource requirement for each real-time flow at a link by exploiting the fact that worst-case delay is rarely experienced by packets traversing a link.

**CDF construction**: Assume that for each packet $k$, the system tracks the run-time measurement history of the ratio $r_k$ of the actual packet delay experienced $D^k_{F,l}$ to the worst-case delay $D^{wc}_{F,l}$, i.e., $r_k = D^k_{F,l}/D^{wc}_{F,l}$ where $r_k$ ranges between 0 and 1. The measured samples of ratio $r_k$ can be used to construct a cumulative distribution function

**Algorithm 4** relax_prob() routine to relax probability assignment vector $\hat{P_F}$, given a delay assignment vector $\hat{D_F}$.

**Algorithm relax_prob($F$, $\hat{D_F}$, $P_F$)**
/* $F$ = Unicast flow being admitted. */
/* $\hat{D_F}$ = Delay assignment vector for flow $F$. */
/* $P_F$ = End-to-end delay violation probability bound. */

$\delta = \frac{1}{2} \min_l \frac{C_l - U_l}{w_l C_l}$;
$b = \delta$;
**while** (1) **do**
  **for** $l = 1$ to $m$ **do**
    $\rho_{F,l} = max\{(C_l - U_l - w_l C_l b),\ \rho_F^{avg}\}$
    $P_{F,l} = \mathcal{P}_l(\rho_{F,l}, D_{F,l})$; /* Violation probability at link $l$ */
  **end for**
  $slack = \frac{\prod_{l=1}^{m}(1 - P_{F,l})}{(1 - P_F)}$;

  **if** ($slack \geq 1$ and $\delta \leq \epsilon$) **then**
    return $\hat{P_F}$;
  **end if**
  $\delta = \delta/2$;

  **if** ($slack > 1$) **then**
    $b = b + \delta$;
  **else**
    $b = b - \delta$;
  **end if**
**end while**

(CDF) $Prob(r)$. Figure 3 shows an example of a CDF constructed in this manner in one simulation instance [14] using aggregate VoIP flows. We can see from the figure that most of the packets experience less than $1/4^{th}$ of their worst-case delay.

**Resource mapping**: The distribution $Prob(r)$ gives the probability that the ratio between the actual delay encountered by a packet and its worst-case delay is smaller than or equal to $r$. Conversely, $Prob^{-1}(p)$ gives the maximum ratio of actual delay to worst-case delay that can be guaranteed with a probability of $p$. The following heuristic gives the correlation function $\mathcal{D}_l(\rho_{F,l}, P_{F,l})$.

$$\mathcal{D}_l(\rho_{F,l}, P_{F,l}) = \left(\frac{L_{max}}{\rho_{F,l}} + \frac{L_{max}}{C_l}\right) \times Prob^{-1}(1 - P_{F,l}) \tag{12}$$

The term $(L_{max}/\rho_{F,l} + L_{max}/C_l)$ represents the deterministic (worst-case) delay from Equation 2 with $\delta_{F,l} = 0$. In other words, to obtain a delay bound of $D_{F,l}$ with a delay violation probability bound of $P_{F,l}$, we need to reserve a minimum bandwidth of $\rho_{F,l}$ which can guarantee a worst-case delay of $D_{F,l}^{wc} = \mathcal{D}_l(\rho_{F,l}, P_{F,l})/Prob^{-1}(1 - P_{F,l})$. The corresponding inverse function $\mathcal{P}_l(\rho_{F,l}, D_{F,l})$ can be derived from Equation 12 as follows.

$$\mathcal{P}_l(\rho_{F,l}, D_{F,l}) = 1 - Prob\left(\frac{D_{F,l}}{\frac{L_{max}}{\rho_{F,l}} + \frac{L_{max}}{C_l}}\right) \tag{13}$$

An important aspect of DDM approach is that the measured CDF changes as new flows are admitted. Hence, before using the distribution $Prob(r)$ to estimate a new flow's resource requirement, DDM needs to account for the new flow's future impact on $Prob(r)$ itself. Details of the impact estimation technique are presented in [14].

# 7 Partitioning for Multicast Flows

In this section, we generalize the algorithms presented in Sections 5 and 6 to multicast flows. We call these algorithms D-MLSA and S-MLSA for deterministic and statistical versions of LSA algorithm for multicast flows.

A real-time multicast flow $M$ consists of a sender at the root and $K$ receivers at the leaves. We assume that the flow requirement is the same value $(D_M, P_M)$ for each of the $K$ leaves, although the number of hops in each of the $K$ paths may be different. A multicast flow $M$ with $K$ receivers can be logically thought of as $K$ unicast flows $F_{M,1} \cdots F_{M,K}$, one flow to each leaf. Although logically treated as separate, the $K$ unicast flows share a single common reservation at each common link along their flow paths. In the following discussion, we overload the notation $M$ to represent the set of links in the multicast tree and overload $F_{M,i}$ to represent the set of links along the $i^{th}$ unicast flow path of the multicast tree.

## 7.1 Admission Control (D-MADM and S-MADM)

A $K$-leaf multicast flow $M$ with an end-to-end delay requirement can be admitted if and only if all of the constituent $K$ (logical) unicast flows $F_{M,1} \cdots F_{M,K}$ can be admitted. The admission control algorithm for multicast flow thus consists of applying variants of the unicast admission control algorithms in Sections 5 and 6 to each of the $K$ unicast paths. The variations account for the fact that the $K$ unicast paths are not completely independent because each unicast flow shares some of the links (and consequently link reservations) with other unicast flows within $M$.

**Algorithm 5** D-MADM: Admission control algorithm for multicast flow $M$ with deterministic delay guarantee $D_M$. The algorithm determines the minimum delay that can be assigned to a multicast flow at each link of a multicast tree.

**Algorithm D-MADM($M$, $D_M$)**
/* $M$ = Multicast flow being admitted. */
/* $D_M$ = End-to-end delay bound. */

**for** each link $l \in M$ **do**
  $D_{M,l} = \mathcal{D}_l(C_l - U_l)$;
**end for**
**for** $i = 1$ to $K$ **do**
  **if** ( $\sum_{l \in F_{M,i}} D_{M,l} > D_M$) **then**
    Reject multicast session $M$;
    exit;
  **end if**
**end for**
Accept multicast session $M$;

Algorithms 5 and 6 give the admission control algorithms for multicast flows with deterministic and statistical delay guarantees respectively. There are two specific variations from the unicast version that deserve mention. First, the order of verifying admissibility among the $K$ unicast paths makes a difference in the case of statistical delay guarantees. Intuitively, more "loaded" paths should be processed earlier since they provide least partitioning flexibility. Hence, we use the sum of the minimum delay budgets $\mathcal{D}_l(C_l - U_l, 0)$ to determine the processing order of the $K$ unicast paths. In other words, paths with higher $\sum_{l \in F_{M,i}} \mathcal{D}_l(C_l - U_l, 0)$ are processed earlier. Secondly,

**Algorithm 6** S-MADM: Admission control algorithm for multicast flow $F_M$ with statistical delay guarantee $(D_M, P_M)$. The algorithm determines the minimum delay and violation probability that can be assigned to a multicast session at each link of a multicast tree.

**Algorithm S-MADM($M$, $D_M$, $P_M$)**
/* $M$ = Multicast flow being admitted. */
/* $D_M$ = End-to-end delay bound. */
/* $P_M$ = End-to-end delay violation probability bound. */

**for** each link $l \in M$ **do**
  $P_{M,l} = 0$;
  $D_{M,l} = \mathcal{D}_l(C_l - U_l, 0)$;
**end for**

**for** each unicast flow $F_{M,i}$ in decreasing order of $\sum_{l \in F_{M,i}} \mathcal{D}_l(C_l - U_l, 0)$ **do**

  **while** $\left( \sum_{l \in F_{M,i}} D_{M,l} > D_M \right)$ and $\left( \prod_{l \in F_{M,i}} (1 - P_{M,l}) \geq (1 - P_M) \right)$ **do**
    $j$ = index of the link in $F_{M,i}$ such that reduction in delay $D_{M,j} - \mathcal{D}_j(C_j - U_j, P_{M,j} + \delta)$ is maximum among all links in the path $F_{M,i}$;

    $P_{M,j} = P_{M,j} + \delta$;

    $D_{M,j} = \mathcal{D}_j(C_j - U_j, P_{M,j})$;
  **end while**

  **if** ( $\sum_{l \in F_{M,i}} D_{M,l} > D_M$) **then**
    Reject multicast session $M$;
    exit;
  **end if**
**end for**
Accept multicast session $M$;

since the $K$ unicast paths are part of the same tree, several of these paths share common links. Before verifying the admissibility of $F_{M,i}$ unicast path, the $D_{M,l}$ values on some of its links may have already been computed while processing unicast paths $F_{M,1}$ to $F_{M,i-1}$. Hence, we carry over the previous $D_{M,l}$ assignments for links that are shared with already processed paths.

Algorithm 5 has a complexity of $O(Kh)$ and Algorithm 6 has a complexity of $O(K \log K + Khn)$. $K$ is the number of leaves, $h$ is the height of the tree, and $n$ is the number of iterations of the inner *while* loop in Algorithm 6, $n$ being dependent upon how soon either the end-to-end delay falls below $D_M$ or the delay satisfaction probability falls below $(1 - P_M)$. Since each iteration increments the value of some $P_{M,j}$ by a positive $\delta$, the *while* loop is guaranteed to terminate.

### 7.2 Load-balancing Slack Allocation (D-MLSA and S-MLSA)

To compute the final delay allocation for the links of a $K$-leaf multicast path, we apply variants of the D-LSA and S-LSA algorithms in Algorithms 1 and 3.

Let's start with the deterministic version D-MLSA given in Algorithm 7. D-MLSA essentially applies the unicast delay relaxation (D-LSA) algorithm to the $K$ unicast paths one after another. There are two salient aspects to the algorithm. First, the delay relaxation is applied to the unicast paths $i = 1$ to $K$ in the increasing order of their current slack in delay budget $(D_M - D_{M,s} - \sum_{l \in F_{M,i}} D_{M,l})$, where $D_{M,s}$ is the smoothing delay defined in Equation 1. This processing order ensures that slack allocation along one path of the tree does not violate end-to-end delay along other paths that may have smaller slack. Secondly, when processing unicast path $F_{M,i}$, the delay relaxation only applies to those links which are not shared with paths $F_{M,1}$ to $F_{M,i-1}$, i.e. those links in $F_{M,i}$ whose $D_{M,l}$ values have not yet been determined. Algorithm 7 has a complexity of $O(K \log K + Kh \log 1/\epsilon)$, where $O(K \log K)$ is to sort the $K$ root-to-leaf delay values and $O(Kh \log 1/\epsilon)$ is for $K$ executions of the bisection method in the D-LSA algorithm.

The S-MLSA heuristic algorithm for multicast flows with statistical delay guarantees, given in Algorithm 8, is similar in structure to the unicast version S-LSA in Algorithm 3. Specifically, the S-MLSA algorithm starts with the minimum delay and probability values obtained in the admission control phase and successively relaxes the delay value assuming fixed probability and the probability values assuming fixed delays. The difference with unicast version is that the delay and probability relaxation steps apply to the entire multicast tree rather than one constituent unicast flow at a time. The multicast_relax_delay() and multicast_relax_prob() procedures are given in Algorithms 9 and 10 and are similar in structure and complexity to the deterministic D-MLSA algorithm.

Note that a multicast tree does not need to be balanced for the MLSA algorithm to be applicable, even though our later performance evaluations are conducted for balanced trees. For non-balanced trees, the MLSA is designed such that each interior link of the tree is assigned the tightest delay budget required to satisfy the QoS requirement of that leaf along the sub-tree following link which has the tightest resource requirement. Any additional slack is distributed along the remaining sub-paths of the sub-tree. Along same lines, while MLSA has been presented with the assumption of same QoS for each leaf, it can be easily generalized for non-uniform QoS for different leaves,

with the core delay partitioning algorithm remaining unchanged.

**Algorithm 7** D-MLSA: Load-balancing Slack Allocation algorithm for a multicast flow $M$ with deterministic delay guarantee $D_M$.

**Algorithm D-MLSA($M$, $D_M$)**
/* $M$ = Multicast flow being admitted. */
/* $D_M$ = End-to-end delay guarantee. */

**for** $l \in M$ **do**
  $D_{M,l} = \mathcal{D}_l(C_l - U_l)$;
**end for**
$R = \emptyset$; /*set of links on which slack has been relaxed*/

**for** each unicast flow $F_{M,i}$ in the order of decreasing $(\sum_{l \in F_{M,i}} D_{M,l})$ **do**

  /*extract the next unicast sub-path $F$ to relax*/
  $F = F_{M,i} - R$;

  /*calculate end-to-end delay requirement $D_F$ over unicast sub-path $F$*/
  $D_F = D_M - \sum_{l \in \{F_{M,i} \cap R\}} D_{M,l}$;

  /*relax the delay vector $\hat{D_F}$ over unicast sub-path $F$*/
  $\hat{D_F} =$ D-LSA($F$,$D_F$)

  $R = R \bigcup F$;
**end for**

**Algorithm 8** S-MLSA: Load-balancing Slack Allocation algorithm for multicast flow $M$ with statistical delay guarantee $(D_M, P_M)$.

**Algorithm S-MLSA($M$,$D_M$,$P_M$)**
/* $M$ = Unicast flow being admitted. */
/* $D_M$ = End-to-end delay bound. */
/* $P_M$ = End-to-end delay violation probability. */

Initialize $\hat{D_M}$ and $\hat{P_M}$ with the final $D_{M,l}$ and $P_{M,l}$ values computed from the admission control algorithm S-MADM;
$iterations = 0$;
**do**
  $\hat{D'_M} = \hat{D_M}$; $\hat{P'_M} = \hat{P_M}$;
  $\hat{D_M} =$ multicast_relax_delay($F_M$, $\hat{P_M}$, $D_M$);
  $\hat{P_M} =$ multicast_relax_prob($F_M$, $\hat{D_M}$, $P_M$);
  $iterations = iterations + 1$;
**while**( (($| \hat{D_M} - \hat{D'_M} | > thresh_D$) OR ($| \hat{P_M} - \hat{P'_M} | > thresh_P$)) AND ($iterations < max\_iterations$));

# 8 Performance Evaluation

## 8.1 Earlier Approaches for Comparison

In this section, we evaluate the performance of the proposed LSA and MLSA algorithms for unicast and multicast flows against three other schemes. The first scheme, named Equal Slack Allocation (ESA), is based on the Equal Allocation (EA) scheme proposed in [25]. EA equally partitions the end-to-end delay among the constituent links in

**Algorithm 9** The multicast_relax_delay() algorithm for a multicast flow $M$ with statistical delay guarantee $(D_M, P_M)$.

**Algorithm multicast_relax_delay($M$, $\hat{P_M}$, $D_M$)**
/* $M$ = Multicast flow being admitted. */
/* $\hat{P_M}$ = Current probability partition vector. */
/* $D_M$ = End-to-end delay. */

$R = \emptyset$; /*set of links on which slack has been relaxed*/

**for** each unicast flow $F_{M,i}$ in the order of decreasing $(\sum_{l \in F_{M,i}} D_{M,l})$ **do**

  /*extract the next unicast sub-path $F$ to relax*/
  $F = F_{M,i} - R$;

  /*calculate end-to-end delay requirement $D_F$ over unicast sub-path $F$*/
  $D_F = D_M - \sum_{l \in \{F_{M,i} \cap R\}} D_{M,l}$;

  /*relax the delay vector $\hat{D_F}$ over unicast sub-path $F$*/
  $\hat{D_F}$ = unicast_relax_delay($F$,$\hat{P_M}$,$D_F$)

  $R = R \bigcup F$;
**end for**

**Algorithm 10** The multicast_relax_prob() algorithm for a multicast flow $M$ with statistical delay guarantee $(D_M, P_M)$.

**Algorithm multicast_relax_prob($M$, $\hat{D_M}$, $P_M$)**
/* $M$ = Multicast flow being admitted. */
/* $\hat{D_M}$ = Current delay partition vector. */
/* $P_M$ = End-to-end delay violation probability. */

$R = \emptyset$; /*set of links on which slack has been relaxed*/

**for** each unicast flow $F_{M,i}$ in the order of increasing $(\prod_{l \in F_{M,i}} (1 - P_{M,l}))$ **do**

  /*extract the next unicast sub-path $F$ to relax*/
  $F = F_{M,i} - R$;

  /*calculate end-to-end delay violation probability requirement $P_F$ over unicast sub-path $F$*/
  $P_F = 1 - \frac{(1-P_M)}{\prod_{l \in \{F_{M,i} \cap R\}} (1-P_{M,l})}$;

  /*relax the probability vector $\hat{P_F}$ over unicast sub-path $F$*/
  $\hat{P_F}$ = unicast_relax_prob($F$, $\hat{D_M}$, $P_F$)

  $R = R \bigcup F$;
**end for**

the path of a unicast flow. As discussed in Section 3, ESA is an improvement over EA since it partitions the slack in end-to-end QoS (delay and/or delay violation probability) equally among the constituent links. MESA is a multicast version of ESA in which the variations proposed in Section 7 are applied. The second scheme, named Proportional Slack Allocation (PSA), is based on the Proportional Allocation (PA) scheme proposed in [8]. PA directly partitions the end-to-end QoS in proportion to loads on constituent links of unicast/multicast path. As with ESA scheme, PSA is a variant of PA that partitions the slack in end-to-end QoS in proportion to the loads on constituent links and MPSA is a multicast variant of PSA.

The third scheme is the Binary-OPQ (or OPQ for short) proposed in [21] for unicast paths, which requires that the global optimization objective be expressed as the sum of per-link cost functions. The cost function that we use for OPQ is is the squared sum of per-link utilization, i.e., goal of partitioning algorithm is to minimize

$$\sum_{l=1}^{m}(U_l/C_l)^2$$

where $U_l$ is the reserved capacity on link l at any time (including the new reservation being admitted), and $C_l$ is the capacity of link $l$. Thus we defined the per-link cost in OPQ as $(U_l/C_l)^2$. The rationale for choosing this function is to try to balance the utilization across all links in the path. The use of squared sum has the effect of minimizing both the magnitude as well as variation of utilization values across different links. Through this cost-function, we attempt to capture the same optimization objective in OPQ as the explicit load balancing criterion in LSA. In fact we also experimented with other cost functions such as $\sum_l(1/(C_l - U_l))^2$. However we found that the use of the former cost function yielded better results (in terms of number admitted flows) for OPQ algorithm. As with LSA, in OPQ we partition the slack in end-to-end QoS rather than the end-to-end QoS directly. MOPQ is the multicast version of OPQ proposed in [21].

Since OPQ and MOPQ operate with only a single end-to-end QoS requirement, we compare them only against the deterministic D-LSA and D-MLSA versions. On the other hand, it is straightforward to extend ESA, PSA and their multicast versions to handle the two simultaneous QoS requirements of end-to-end delay and delay violation probability. Hence we compare these schemes for both deterministic and statistical cases. As a note on terminology, D-ESA and S-ESA refer to deterministic and statistical versions of ESA scheme for unicast flows, D-MESA and S-MESA refer to the same for multicast flows and so on for PSA.

The admission control algorithm that we use for ESA, PSA, OPQ, and their multicast versions is exactly the same as what we propose in this paper for LSA and MLSA. In other words, before slack allocation is performed using any of the schemes, the decision on whether to admit a new flow is made by comparing the accumulated end-to-end minimum QoS against the required end-to-end QoS. Thus the differences shown in performance result solely from different techniques for slack allocation.

### 8.2 Evaluation Setup

We evaluate the performance of different slack allocation algorithms using unicast paths, multicast trees, and general network topologies. The first topology for unicast paths (*Unicast-1*) has 45 Mbps capacity at the last link and

90 Mbps capacity at all other links. The second topology for unicast paths (*Unicast-2*) has a random mix of link capacities between 45 Mbps to 200 Mbps. Similarly, the *Multicast-1* topology consists of a tree in which destinations are connected to 45 Mbps links whereas interior links have 90 Mbps capacity. The *Multicast-2* topology consists of a random mix of link capacities between 45 Mbps to 200 Mbps. Unicast algorithms are also compared using $5 \times 5$ and $6 \times 6$ *Grid* topologies and *Sprint's North American IP Backbone* topology, which consists of OC-192 bidirectional links. For the simulations, we select the link capacities in both Sprint and Grid topologies from a random mix between 45 Mbps and 200 Mbps. In each simulation instance, sources and destinations are selected uniformly among all nodes that are separated by a fixed shortest path distance. In other words, all source-destination pairs are separated by a path-length of $h$, where $h$ varies from 6 to 10 depending upon the specific experiment. Flows are established along the shortest paths connecting the selected sources and destinations.

Evaluations of LSA algorithms on unicast and multicast paths use $w_l = 1$ in the slack allocation phase since it leads to perfect load balancing. On the other hand, with network-wide load-balancing as the optimization criteria for the Grid and Sprint network topologies, we select $w_l = 1/u_l$, where $u_l$ is the *expected utilization* for link $l$ obtained by means of traffic profiling. In other words, $w_l$ values are set in inverse proportion to expected link utilization values $u_l$. For general network topologies, other values of $w_l$ may be chosen depending upon the optimization objective of the routing algorithms being employed in the network. The traffic profile is generated by initially admitting flow requests based on their long-term average rates till the network is saturated to capacity. During the actual execution of LSA, flow requests are generated in the same order as during profile generation, but the admission decisions are now based on both delay constraints as well as average rate constraints.

Algorithms for deterministic end-to-end delay guarantees (D-* schemes) are evaluated using C++ implementations whereas those for statistical delay guarantees (S-* schemes) are evaluated using dynamic trace driven simulations with the ns-2 network simulator. Each real-time flow traffic in trace driven simulations consists of aggregated traffic traces of recorded VoIP conversations used in [17], in which spurt-gap distributions are obtained using G.729 voice activity detector. Each VoIP stream has an average data rate of around 13 kbps, peak data rate of 34 kbps, and packet size of $L_{max} = 128$ bytes. We temporally interleave different VoIP streams to generate 5 different aggregate traffic traces, each with a data rate of $\rho^{avg} = 100$kbps. Each aggregated flow trace is 8073 seconds long. Every real-time flow sends traffic for the entire lifetime of the simulation with the aggregate traffic trace being repeated over its lifetime. The results shown in statistical experiments are average values over 10 test runs with different random number seed used to select VoIP traces and initiate new flows. Flow requests arrive with a random inter-arrival time between 1000 to 5000 seconds. We perform evaluations mainly for the 'static' case in which flow reservations that are provisioned once stay in the network forever. CDF used in admission control decisions at each link is measured over the time-interval between flow arrivals. The WFQ [35] service discipline is employed for packet scheduling at each link in order to guarantee the bandwidth shares of flows sharing the same link.

In the rest of the section, we present performance results for unicast topology with deterministic and statistical delay guarantees, and for multicast trees and general network topologies with deterministic delay guarantees. For multicast trees and general network topologies, the memory requirements in the case of statistical trace driven

simulations do not scale in our current system.

### 8.3 Effectiveness of LSA Algorithm

We first take a snapshot view of the performance of LSA algorithm in comparison to ESA, PSA and OPQ algorithms and later examine the impact of different parameters in detail. Table 1 shows the number of flows admitted over 7-hop paths with deterministic and statistical delay requirements. Figure 4 plots the number of flows admitted with deterministic delay guarantees under Unicast-2, Multicast-2, 5x5 Grid and Sprint topologies for different mixes of link bandwidths. The table and figures demonstrate that in all scenarios, LSA consistently admits more number of flows than all other algorithms. This is because LSA explicitly attempts to balance the loads across different links. In contrast, ESA algorithm does not optimize any specific metric and PSA algorithm does not explicitly balance the loads among the links. Similarly we see that the performance obtained with OPQ algorithm is worse than that with LSA.

*The main problem does not lie within the OPQ algorithm itself, but in coming up with a cost metric that accurately captures the load balancing criteria.* In this case, OPQ algorithm performs its work of coming up with a solution that is close to optimal in minimizing the specific cost metric; however, the best cost-metric we can construct turns out to be only an approximation of the final load balancing optimization objective. Instead of implicitly capturing the load balancing criteria by means of a cost function, the LSA algorithm approaches the problem in a reverse fashion by exploring only those slack partitions that maintain the required load balance among the links.

### 8.4 Capacity Evolution

In order to understand why the LSA algorithm admits a higher number of flow requests than other algorithms, we compare their resource usage patterns. Figure 5 plots the evolution of available link bandwidth on the each constituent link of the Unicast-2 topology when flows require a deterministic end-to-end delay bound of $60$ms. Each curve corresponds to one link in the unicast path. Figure 6 plots the same curves for each link when flows require statistical end-to-end delay bound of $45$ms and delay violation probability of $10^{-5}$. We used a smaller statistical delay bound of $45$ms compared to the deterministic delay bound of $60$ms in Figure 5 because, for statistical case, the difference in performance among different algorithms is evident only at smaller delay bounds. Figure 7 plots the same curves for Multicast-2 topology with tree depth of 7 when multicast flows require end-to-end deterministic delay bound of $60$ms. Each curve corresponds to one link in the multicast path.

Note that, no more flows can be admitted along the unicast/multicast path after the residual capacity of any one link falls to zero, even though other links may still have unused capacity. Thus a single link with insufficient residual capacity is enough to render the entire network path unusable for newer reservations. In each figure, maximum number of flows are admitted in the case of *-LSA algorithm where a high degree of load balance is maintained among all links with every admitted flow. On the other hand, the link loads are more imbalanced in the case of *-ESA, *-PSA and *-OPQ algorithms, as evident from the more widely spaced curves for different links. Specifically, the bandwidth of the links with lower capacity is consumed more quickly than that of links with higher capacity

and consequently fewer number of flows are admitted. Among ESA, PSA and OPQ algorithms, OPQ and PSA have similar performance followed by the ESA. The differences in performance arise from the extent to which each algorithm accounts for load-imbalance between links. For multicast flows in Figure 7, the capacity evolution curves are not perfectly balanced in the case of D-MLSA due to the fact that the assigned bandwidth on some of the links is lower-bounded by the 100 Kbps average rate of flows which is larger than the 'delay-derived' bandwidth required to satisfy the delay budget at those link.

### 8.5 Effect of End-to-end Delay

Figure 8 plots the variation in number of admitted flows over unicast and multicast topologies as their end-to-end deterministic delay requirement is varied. With increasing delay, all the four algorithms admit more number of flows, since a less strict end-to-end delay bound translates to lower resource requirement at intermediate links. Again, LSA admits more flows than others since it maintains load-balance while partitioning the slack. A maximum of 450 flows with 100 Kbps average data rate can be admitted by any of the algorithms in Unicast-2 and Multicast-2 topologies because the smallest link capacity is 45 Mbps.

### 8.6 Effect of End-to-end Delay Violation Probability

Figure 9 plots the variation in average number of admitted flows over unicast and multicast paths as their delay violation probability bound is varied from $10^{-6}$ to $10^{-1}$ for a $45$ms end-to-end delay bound. The LSA algorithm is able to admit far more flows than ESA and PSA algorithms since it can partition the slack in a load-balancing manner along both the dimensions of delay as well as delay violation probability. The performance gain for LSA over ESA and PSA is much larger than in the case of deterministic delay requirements (Figure 8) because even a small increase in delay violation probability yields a significant reduction in resources assigned to a flow.

### 8.7 Effect of Path Length

Figure 10 plots the variation in number of admitted flows having deterministic delay requirements of $60$ms, as the path length/tree depth increases. For all the four algorithms, there is a drop in the number of flows admitted with increasing path length because the same end-to-end delay now has to be partitioned among more number of intermediate hops. Thus increasing the path length has the effect of making the end-to-end requirement more strict, which in turn translates to higher resource requirement at the intermediate links. For Unicast-2 and Multicast-2 topologies, the LSA algorithm still outperforms the other three algorithms in terms of number of flows it can support since it better manages the decreasing slack in delay to counter load imbalance among links. For Grid and Sprint topologies, performance of LSA is similar to ESA algorithm for small paths lengths (less than 5); PSA and OPQ perform worse than both ESA and LSA for small path lengths. This is because at smaller path lengths, flow reservations are mainly guided by their average rate constraint rather than delay constraints and using ESA turns out to be more prudent in a general network context. However, with increasing path lengths (larger than 5), LSA admits

up to 12% more flows than PSA, which performs the next best. This is because at beyond path length of 5, flow reservations are guided by their delay constraints (rather than average rate) and the benefits of LSA become evident.

# 9 Conclusion

Resource provisioning techniques for network flows with end-to-end delay guarantees need to address an intra-path load balancing problem such that none of the constituent links of a selected path exhausts its capacity long before others. By avoiding such load imbalance among the links of a path, resource fragmentation is less likely and more flows can be admitted in the long term. We have proposed a set of load-aware delay budget partitioning algorithms that can solve this intra-path load balancing problem for both unicast and multicast flows with either deterministic or statistical delay requirements. In particular, the proposed Load-balancing Slack Allocation (LSA) algorithms allocate a larger share of the delay slack to more loaded links than to less loaded links, thus reducing the load deviation among these links. Through a detailed comparison with three other algorithms, we have shown that the LSA algorithms can indeed improve the number of admitted flows by up to 1.2 times for deterministic delay bounds and 2.8 times for statistical delay bounds.

In a larger context, the proposed delay partitioning algorithm is just one component of a comprehensive network resource provisioning framework. While the proposed algorithms are described in the context of a given unicast paths and multicast trees, eventually these algorithms need to be integrated more closely with other components such as QoS-aware routing or inter-path load balancing to achieve network-wide optimization. Quantitatively exploring the inherent interactions between intra-path and inter-path load balancing schemes is a fruitful area for future research.

# Acknowledgment

We would like to thank Henning Schulzrinne and Wenyu Jiang for providing the VoIP traces used in our simulations. We would also like to thank Pradipta De, Ashish Raniwala, and Srikant Sharma for their insightful suggestions that helped improve the presentation of this paper. Based upon work supported in part by National Science Foundation under Grants CNS-0509131, CCF-0541096, ACI-0234281, and CNS-0435373.

# References

[1] L. Andersson, N. Feldman, A. Fredette, and B. Thomas. *Label Distribution Protocol Specification*. RFC 3036, Jan. 2001.

[2] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proc. of ACM Sigcomm'03*, October 2003.

[3] D.O. Awduche, L. Berger, D. Gan, T.Li, G.Swallow, and V.Srinivasan. *Extensions to RSVP for LSP Tunnels*. RFC 3209, Dec. 2001.

[4] E. Biton and A. Orda. Qos provision and routing with stochastic guarantees. In *Proc. of QShine*, 2004.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queuing algorithm. In *Proc. of ACM SIGCOMM'89*, pages 3–12, 1989.

[6] F. Ergun, R. Sinha, and L. Zhang. QoS routing with performance dependent costs. In *Proc. of INFOCOM 2000, Tel Aviv, Israel*, March 2000.

[7] D. Ferrari and D.C. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3):368–379, April 1990.

[8] V. Firoiu and D. Towsley. Call admission and resource reservation for multicast sessions. In *Proc. of IEEE INFOCOM'96*, 1996.

[9] L. Georgiadis, R. Guerin, V. Peris, and K. N. Sivarajan. Efficient network QoS provisioning based on per node traffic shaping. *IEEE/ACM Transactions on Networking*, 4(4):482–501, August 1996.

[10] K. Gopalan. *Efficient Provisioning Algorithms for Network Resource Virtualization with QoS Guarantees*. PhD thesis, Computer Science Dept., Stony Brook University, August 2003.

[11] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *Proc. of Multimedia Computing and Networking 2002, San Jose, CA, USA*, pages 34–45, Jan. 2002.

[12] K. Gopalan, T. Chiueh, and Y.J. Lin. Network-wide load balancing routing with performance guarantees. In *Proc. of ICC 2006, Istanbul, Turkey*, June 2006.

[13] K. Gopalan, T. Chiueh, and Y.J. Lin. Delay budget partitioning for delay bounded network paths. In *Proc. of Infocom 2004, Hong Kong, China*, March 2004.

[14] K. Gopalan, T. Chiueh, and Y.J. Lin. Probabilistic delay guarantees using delay distribution measurement. In *Proc. of ACM Multimedia 2004*, pages 900–907, New York, NY.

[15] R. Guerin and A. Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. *IEEE/ACM Transactions on Networking*, 7(3):350–364, June 1999.

[16] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches*. MIT Press, 1988.

[17] W. Jiang and H. Schulzrinne. Analysis of On-Off patterns in VoIP and their effect on voice traffic aggregation. In *Proc. of ICCCN 2000*, March 1996.

[18] M. Kodialam, T. V. Lakshman, and S. Sengupta. Efficient and robust routing of highly variable traffic. In *Proc. of HotNets III*, November 2004.

[19] M. Kodialam and S.H. Low. Resource allocation in a multicast tree. In *Proc. of INFOCOM 1999, New York, NY*, March 1999.

[20] J. Liebeherr, D.E. Wrege, and D. Ferrari. Exact admission control for networks with a bounded delay service. *IEEE/ACM Transactions on Networking*, 4(6):885–901, Dec. 1996.

[21] D.H. Lorenz and A. Orda. Optimal partition of QoS requirements on unicast paths and multicast trees. *IEEE/ACM Transactions on Networking*, 10(1):102–114, February 2002.

[22] D.H. Lorenz and A. Orda. QoS routing in networks with uncertain parameters. *IEEE/ACM Transactions on Networking*, 6(6):768–778, December 1998.

[23] D.H. Lorenz, A. Orda, and D. Raz. Optimal partition of QoS requirements for many to many connections. In *Proc. of INFOCOM'03*, March 2003.

[24] Q. Ma and P. Steenkiste. Quality of service routing for traffic with performance guarantees. In *Proc. of IWQoS*, 1997.

[25] R. Nagarajan, J. Kurose, and D. Towsley. Local allocation of end-to-end quality-of-service in high-speed networks. In *Proc. of 1993 IFIP Workshop on Perf. analysis of ATM Systems, North Holland*, pages 99–118, Jan. 1993.

[26] A.K. Parekh. *A generalized processor sharing approach to flow control in integrated services networks*. PhD thesis, Massachusetts Institute of Technology, Feb. 1992.

[27] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case. *IEEE/ACM Transactions on Networking*, 2(2):137–152, April 1994.

[28] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.

[29] D. Raz and Y. Shavitt. Optimal partition of QoS requirements with discrete cost functions. In *Proc. of INFOCOM 2000, Tel Aviv, Israel*, March 2000.

[30] M. Reisslein, K.W. Ross, and S. Rajagopal. A framework for guaranteeing statistical QoS. *IEEE/ACM Transactions on Networking*, 10(1):27–42, February 2002.

[31] A. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. RFC 2212, Sept. 1997.

[32] K.A. Sikorski. *Optimal Solution of Nonlinear Equations, ISBN 0195106903*. Oxford University Press, 2001.

[33] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proc. of IEEE*, 83(10):1374–1396, October 1995.

[34] H. Zhang and E. Knightly. Providing end-to-end statistical performance guarantees with bounding interval dependent stochastic models. In *Proc. of ACM SIGMETRICS'94, Nashville, TN*, pages 211–220, May 1994.

[35] L. Zhang. Virtual Clock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems*, 9(2):101–124, May 1991.

[36] R. Zhang-Shen and N. McKeown. Designing a predictable internet backbone with valiant load-balancing. In *Proc. of Intl. Workshop on Quality of Service (IWQoS 2005), Passau, Germany*, June 2005.

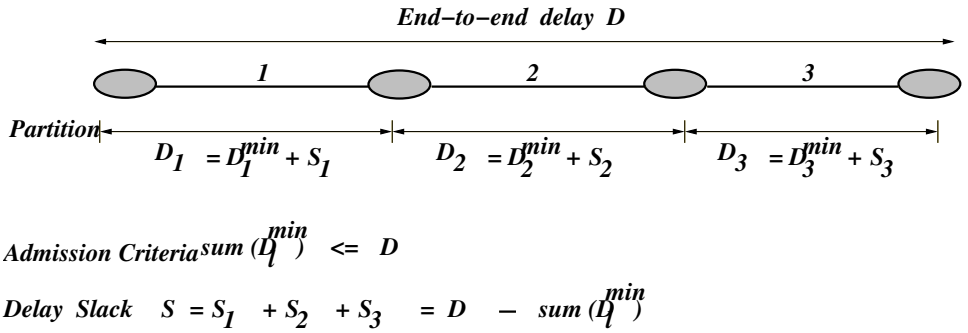


Figure 1: Example of partitioning end-to-end delay budget $D$ over a three-hop path. The slack $S$ indicates the amount of flexibility available in partitioning the delay budget $D$.

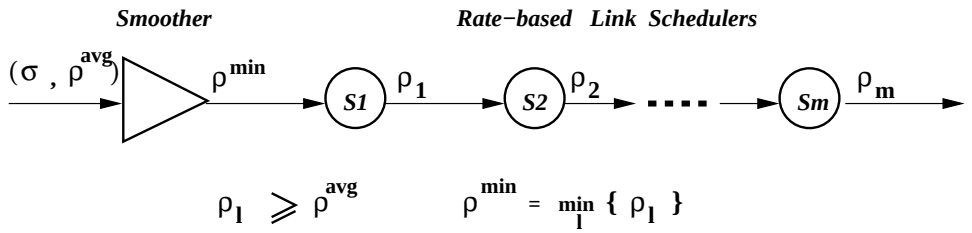


Figure 2: Network components along a multi-hop flow path. Flow $F$ that has $(\sigma, \rho^{avg})$ input traffic characterization passes through a smoother followed by a set of rate-based link schedulers. $F$'s service rate at each link $l$ is $\rho_l \geq \rho^{avg}$ and the bursts are smoothed before the ingress at a rate of $\rho^{min} = \min_l\{\rho_l\}$.

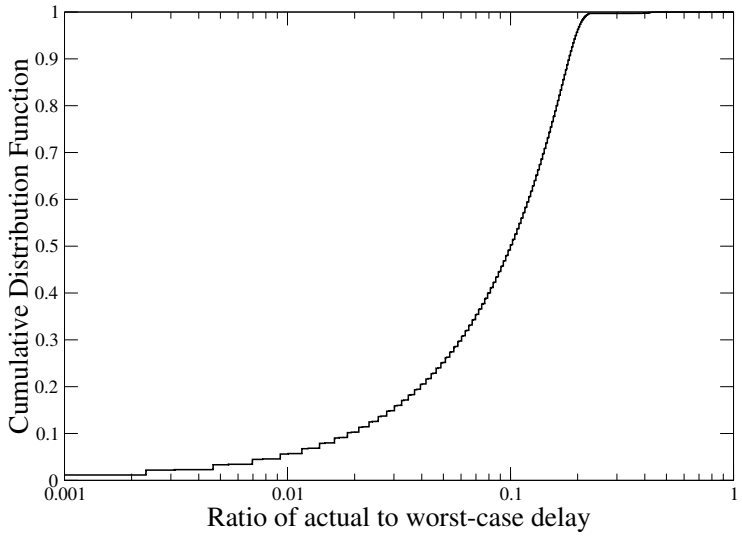


Figure 3: Example of cumulative distribution function (CDF) of the ratio of actual delay to worst-case delay experienced by packets. X-axis is in log scale.

| **Scenario** | Topology | **ESA** | **PSA** | **OPQ** | **LSA** |
|---|---|---|---|---|---|
| Unicast/Deterministic | Unicast-1 | 219 | 263 | 271 | 295 |
| $D = 60$ms | Unicast-2 | 225 | 275 | 284 | 310 |
| Unicast/Statistical | Unicast-1 | 81 | 121 | N/A | 319 |
| $D = 45$ms $P = 10^{-5}$ | Unicast-2 | 81 | 117 | N/A | 292 |
| Multicast/Deterministic | Multicast-1 | 221 | 268 | 265 | 292 |
| $D = 60$ms | Multicast-2 | 219 | 249 | 244 | 267 |
| Grid/Deterministic | 5x5 | 2906 | 3237 | 3179 | 3495 |
| $D = 60$ms | 6x6 | 6280 | 6362 | 6386 | 6983 |
| Sprint/Deterministic | US IP | 1320 | 1585 | 1561 | 1614 |
| $D = 60$ms | Backbone | | | | |

Table 1: Flows admitted with ESA, PSA, OPQ, and LSA algorithms. Hops=7, $\rho^{avg} = 100Kbps$.

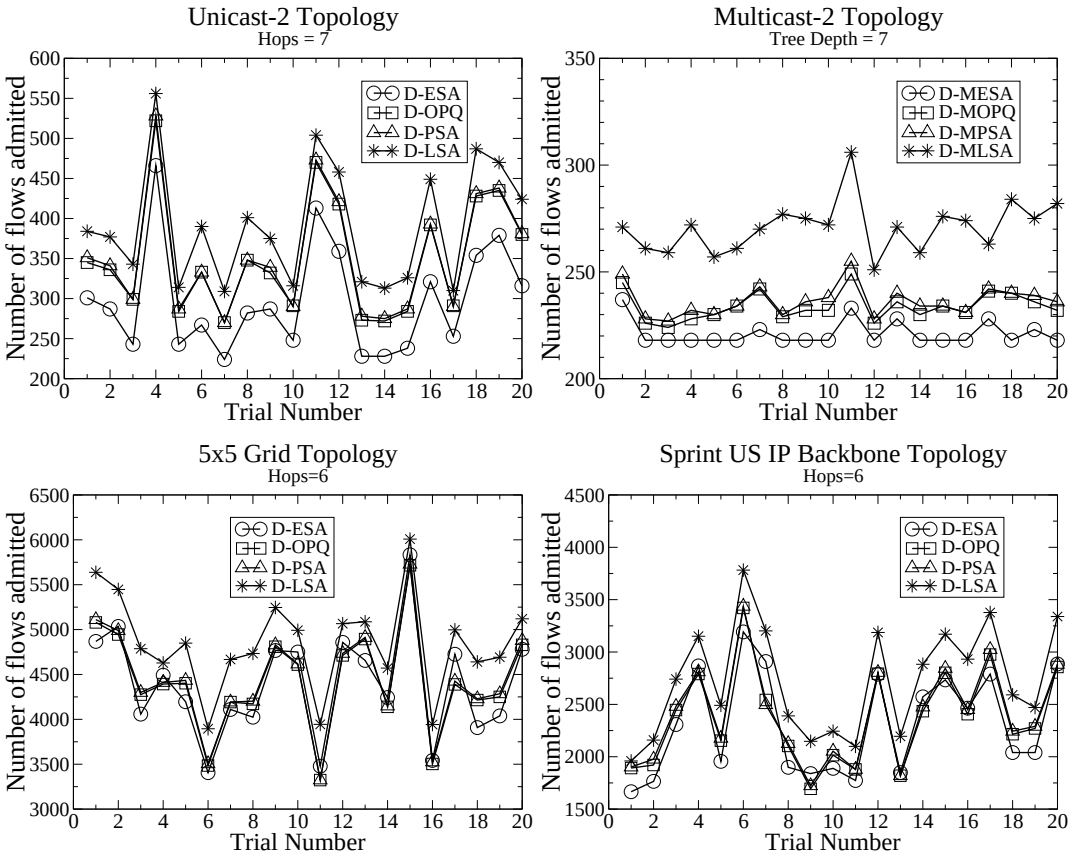


Figure 4: Number of flows admitted with for deterministic delay guarantees. $\rho^{avg} = 100Kbps$, $D = 60ms$.

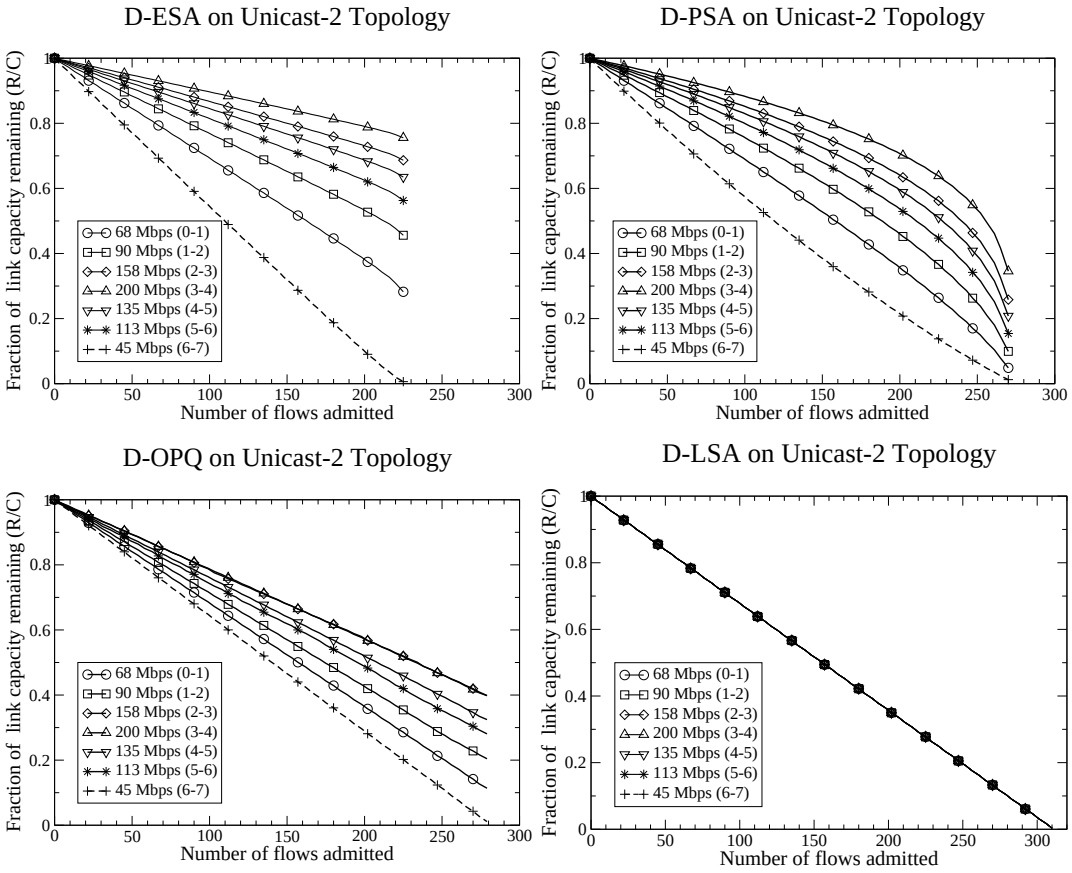


Figure 5: Evolution of available link capacity with the number of flows admitted for unicast flows with deterministic delay guarantee. Hops=7, $\rho^{avg} = 100Kbps$, $D = 60ms$. Each curve within a graph corresponds to one link in the unicast path. Each graph corresponds to one slack partitioning scheme (D-ESA, D-PSA, D-OPQ, or D-LSA). The residual capacity of each link can reduce at a different rate with each admitted flow. Closely spaced curves correspond to better load balance among links and more number of admitted flows than widely spaced curves.

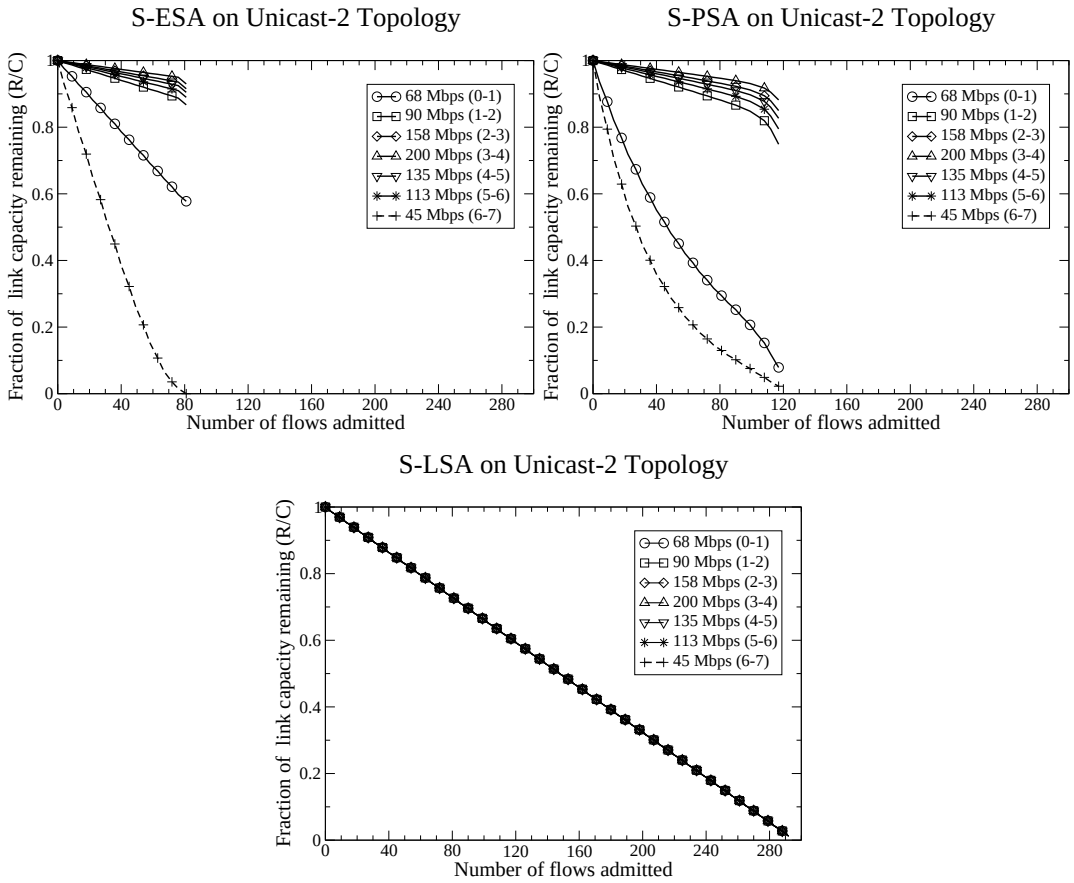


Figure 6: Evolution of available link capacity with the number of flows admitted for unicast flows with statistical delay guarantee. Hops=7, $\rho^{avg} = 100Kbps$, $D = 45ms$, $P = 10^{-5}$. Each curve within a graph corresponds to one link in the unicast path. Each graph corresponds to one slack partitioning scheme (S-ESA, S-PSA, S-OPQ, or S-LSA). The residual capacity of each link can reduce at a different rate with each admitted flow. Closely spaced curves correspond to better load balance among links and more number of admitted flows than widely spaced curves.

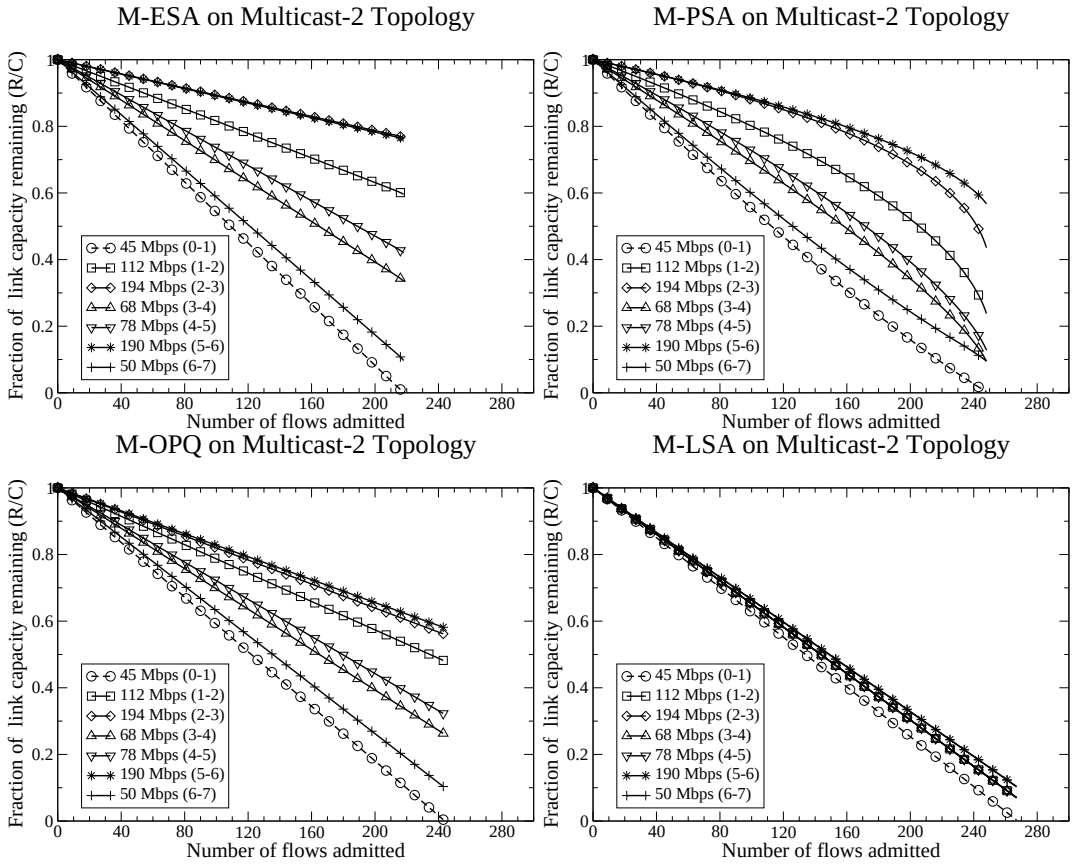


Figure 7: Evolution of available link capacity with the number of flows admitted for multicast flows with deterministic delay guarantee. Tree depth=7, $\rho^{avg} = 100Kbps$, $D = 60ms$. Each curve within a graph corresponds to one link in the multicast path. Each graph corresponds to one slack partitioning scheme (M-ESA, M-PSA, M-OPQ, or M-LSA). The residual capacity of each link can reduce at a different rate with each admitted flow. Closely spaced curves correspond to better load balance among links and more number of admitted flows than widely spaced curves.

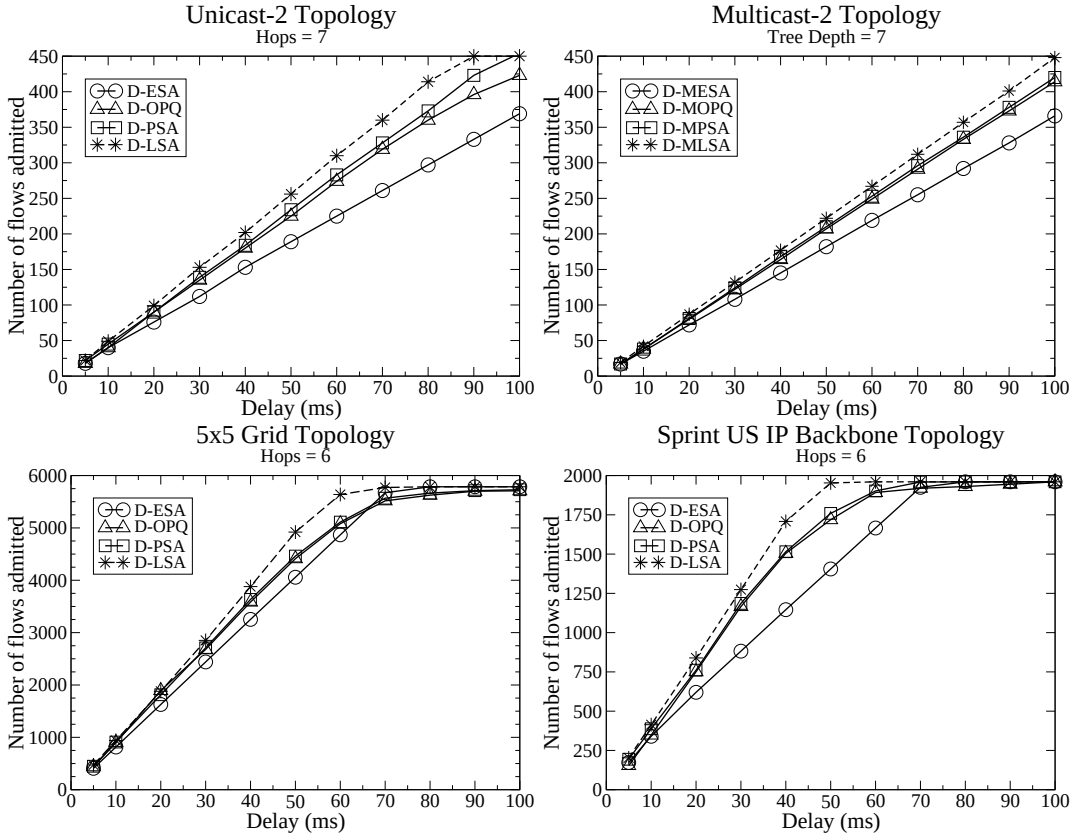


Figure 8: Number of flows admitted vs. deterministic end-to-end delay bound. $\rho^{avg} = 100Kbps$.

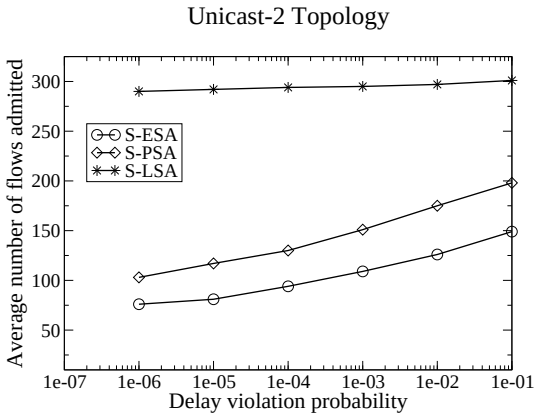


Figure 9: Average number of flows admitted vs. end-to-end delay violation probability bound. Hops=7, $D = 45ms$, $\rho^{avg} = 100Kbps$. Averages computed over ten simulation runs with different random seeds.

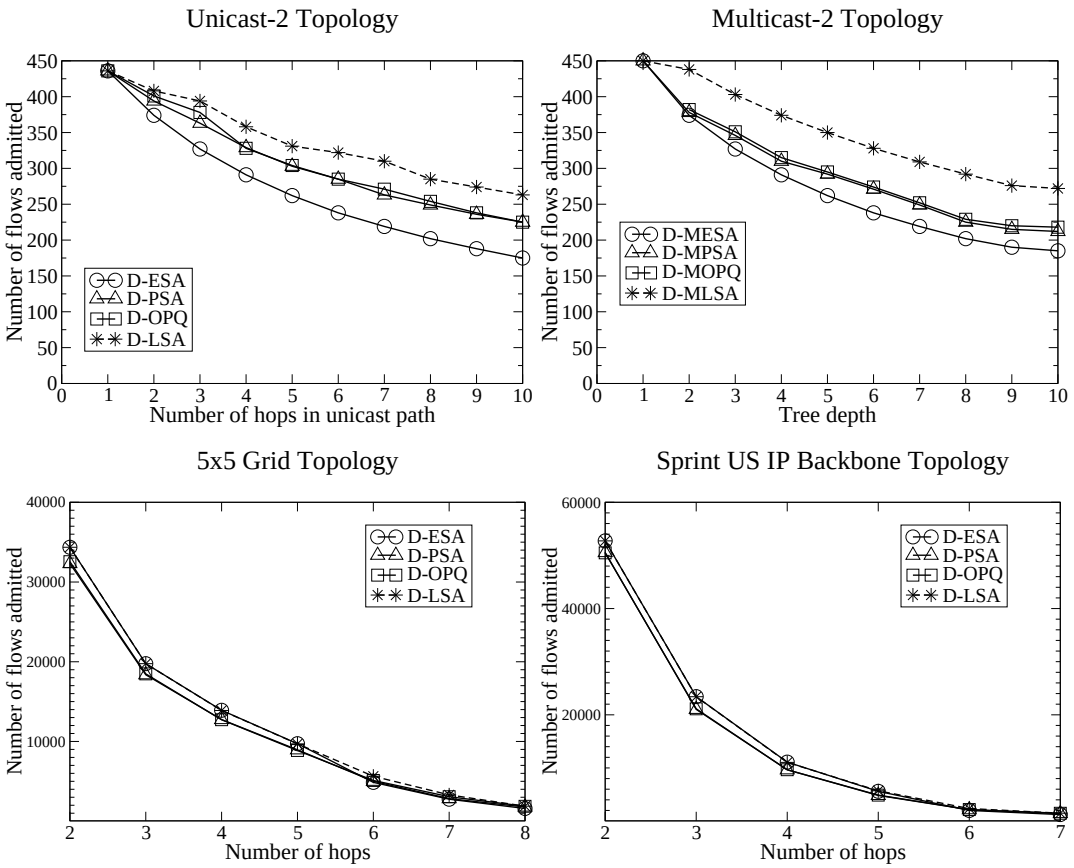


Figure 10: Number of flows admitted vs. path length. $D = 60ms$, $\rho^{avg} = 100Kbps$.