Delay Budget Partitioning to Maximize Network Resource Usage Efficiency

Kartik Gopalan
Florida State University
kartik@cs.fsu.edu

Tzi-cker Chiueh
Stony Brook University
chiueh@cs.sunysb.edu

Yow-Jian Lin
Telcordia Technologies
yjlin@research.telcordia.com

***Abstract*— Provisioning techniques for network flows with end-to-end QoS guarantees need to address the inter-path and intra-path load balancing problems to maximize the resource utilization efficiency. This paper focuses on the intra-path load balancing problem: How to partition the end-to-end QoS requirement of a 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 new algorithm to solve the end-to-end QoS partitioning problem for unicast and multicast flows that takes into account the loads on the constituent links of the chosen flow path. This algorithm can simultaneously partition multiple end-to-end QoS requirements such as the end-to-end delay and delay violation probability bound. The key concept in our proposal is the notion of *slack*, which quantifies the extent of flexibility available in partitioning the end-to-end delay requirement across the links of a selected path (or a multicast 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. A detailed simulation study demonstrates that, compared with previous approaches, the proposed delay budget partitioning algorithm 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.**

I. Introduction

Performance-centric network applications such as Voice over IP (VoIP), video conferencing, streaming media and online trading have stringent Quality of Service (QoS) requirements in terms of end-to-end delay and throughput. In order to provide QoS guarantees, the network service provider needs to dedicate part of network resources for each customer. Hence an important problem faced by every provider is *how to maximize the utilization efficiency of its physical network infrastructure and still support heterogeneous QoS requirements of different customers*. Here utilization efficiency is measured by the amount of customer traffic supported over a fixed network infrastructure.

Traffic engineering techniques in Multi-Protocol Label Switched (MPLS) networks select explicit routes for Label Switched Paths (LSP) between a given source and destination. Each LSP could act as a traffic trunk carrying an aggregate traffic flow that requires QoS guarantees such as bandwidth and delay bounds. In our terminology, a flow represents a long-lived aggregate of network connections (such as a VoIP trunk) rather than short-lived individual streams (such as a single VoIP conversation). The key approach underlying traffic engineering algorithms, such as [1], is to select network paths so as to balance the loads on the network links and routers. 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 real-time network flows that require end-to-end delay guarantees, there is an additional optimization dimension for balancing loads on the network links, namely, the partitioning of end-to-end QoS requirements along the links of a selected network path. 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 [2]. Given a request for setting up a real-time flow F_i along this path with a deterministic end-to-end delay requirement D_i, we need to assign delay budgets $D_{i,l}$ to F_i at each individual link l of the path. Intuitively, low delay typically requires high bandwidth reservation. In other words, since flow F_i's packet delay bound at each link is inversely proportional to its bandwidth reservation, the amount of delay budget allocated to F_i at a link determines the amount of bandwidth reservation that F_i requires at that link.

The question is, *how do we partition the end-to-end delay requirement D_i into per-link delay budgets such that (a) the end-to-end delay requirement D_i 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 tolerate their packets experiencing end-to-end delays in excess of D_i within a small delay violation probability P_i. Such *statistical* delay requirements of the form (D_i, P_i) 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 requirement P_i, in addition to the delay requirement D_i, the delay budget partitioning problem is generalized to statistical delay requirements for network flows.

This paper makes the following main contributions. Firstly, we present a unified algorithm template, called Load-based Slack Sharing (LSS) for the delay budget partitioning problem, and describe its application to unicast and multicast flows with deterministic and statistical delay guarantees. Secondly, our algorithm can handle multiple simultaneous flow QoS require-

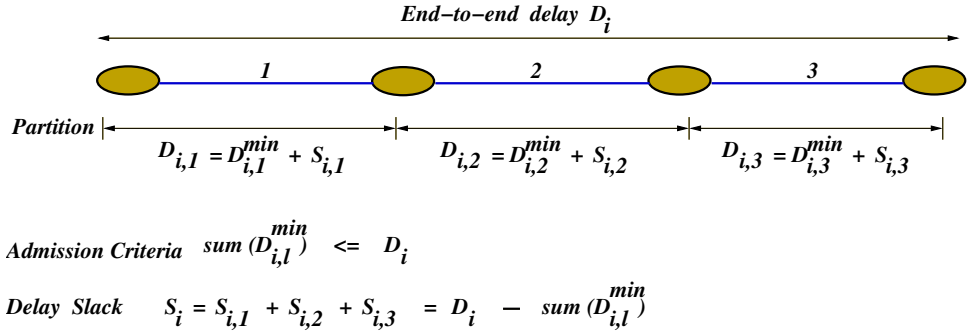

Fig. 1. Example of partitioning end-to-end delay budget D_i over a three-hop path. The slack S_i indicates the amount of flexibility available in partitioning the delay budget D_i.

ments such as end-to-end delay bounds and delay violation probability bounds. Earlier approaches handled only a single end-to-end QoS requirement at a time. Thirdly, 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 II. 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 [3] where the network links are serviced by packetized rate-based schedulers [2] [4] [5] [6] 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 [7]. Note that we address the problem of partitioning *additive* end-to-end delay requirement and the *multiplicative* delay violation probability requirement. Specifically, the problem we address is *not* the same as partitioning *bottleneck* QoS parameters. Indeed, for packetized rate-based schedulers such as WFQ, the end-to-end delay bound for a flow contains both additive (per-link) and bottleneck delay components. In Section IV, 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 one of the components of a larger network resource management system [8] 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.

The rest of the paper is organized as follows. Section II introduces the notion of *slack sharing* which is central to our work. Section III places our work in the context of related research in delay budget partitioning. Section IV formulates our model of the network and reviews standard results for end-to-end delay bounds. In Section V, VI and VII 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 VIII we analyze the performance of the proposed algorithms and Section IX concludes with a summary of main research results.

II. Notion of Slack and Slack Sharing

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 $D_{i,l}^{min}$ guaranteed to a new flow F_i 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_i and the sum of minimum delay requirements $D_{i,l}^{min}$, i.e. $\Delta D_i = D_i - \sum_{l=1}^{3} D_{i,l}^{min}$, represents an excess *slack* that can be shared among the links to reduce their respective bandwidth loads.

The manner in which slack is shared 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 share it based on the current loads on the network links. For example, assume that the three links in Figure 1 carry a load of 40%, 80% and 20% respectively. Given a total slack in delay budget ΔD_i, one could partition the slack proportionally as $\frac{2}{7}\Delta D_i$, $\frac{4}{7}\Delta D_i$, and $\frac{1}{7}\Delta D_i$, respectively, rather than assigning each $\frac{1}{3}\Delta D_i$. 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 scheme, on the other hand, would lead to second 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.

III. Related Work

Equal Allocation (EA) scheme, proposed in [9], divides the end-to-end loss rate requirement equally among constituent links. The principal conclusion in [9] is similar to ours, that is the key to maximize resource utilization is to reduce the load imbalance across network links. The work focused 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. Proportional Allocation (PA) scheme, proposed in [7], considered partition of end-to-end QoS over links of a multicast tree in proportion to the utilization of each link. The performance of PA was shown to be better than EA since it accounts for different link loads. Our work is different from the above two proposals in the following aspects. First, we use a heuristic that directly balances the loads on different links instead of indirectly addressing the objective via equal/proportional allocation. Secondly, instead of partitioning the slack in QoS, the above two proposals partition the entire end-to-end QoS requirement directly among the links. If an equal/proportional partition results in tighter delay requirement than the minimum possible at some link, then the proposal in [7] 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, we partition the slack in end-to-end delay, instead of the end-to-end delay, which helps prevent the formation of bottleneck links as long as non-zero slack is available.

Efficient algorithms to partition the end-to-end QoS requirements of a unicast or multicast flow into per-link QoS requirements have been proposed in [10] [11] [12]. The optimization criteria is to minimize a global cost function which is the sum of local link costs. The cost functions are assumed to be weakly convex in [10] and increase with the severity of QoS requirement at the link whereas [11] addresses general cost functions. On the other hand, [13] addresses the problem in the context of discrete link costs in which each link offers only a discrete number of QoS guarantees and costs. For the algorithms in [10] [11] [12] [13] 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 VIII, the best cost function that we could devise to capture the load-balancing optimization criteria, when used with algorithms in [10], 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 LSS algorithm explores only those QoS partitions that maintain explicit load-balance among links. Furthermore, the above 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.

The problem of partitioning as well as QoS routing has been addressed in [14] [15]. The goal of [14] [15] is different from ours, namely that of maximizing the probability of meeting the QoS requirements of a flow. While we do not address the routing problem in this paper and focus on the intra-path QoS partitioning problem, an approach to integrate our proposal in this paper with QoS routing schemes has been outlined in [8]. 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 [16]. However, equal allocation scheme is used to assign per-link delay and packet loss probability bounds. An approach to provide end-to-end statistical performance guarantees has been proposed in [17] when the traffic sources are modeled with a family of bounding interval-dependent random variables. Rate controlled service discipline is employed inside the network. However, the work does not address the issue of how to locally partition the end-to-end delay requirement.

IV. Network Model

A *real-time flow* F_i is defined as an aggregate that carries traffic with an average bandwidth of ρ_i^{avg} and burst size σ_i. We assume that the amount of flow F_i traffic arriving into the network in any time interval of length τ is bounded by $(\sigma_i + \rho_i^{avg}\tau)$. The (σ_i, ρ_i^{avg}) characterization can be achieved by regulating F_i'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 [18] [5]. Specifically, as shown in Figure 2, a regulated flow F_i 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 smoothes the burstiness in F_i's traffic before the ingress node. Each rate-based scheduler at intermediate link l services the flow at an assigned rate $\rho_{i,l} \geq \rho_i^{avg}$. The manner in which rates $\rho_{i,l}$ are assigned will be described later in Sections V and VI. The smoother regulates flow F_i's traffic at a rate $\rho_i^{min} = \min_l\{\rho_{i,l}\}$ i.e., at the smallest of per-link assigned rates. Since flow F_i's service rate at each link scheduler is greater or equal to smoothing rate at the ingress, F_i's traffic does not become bursty at any of the intermediate links. Completely smoothing F_i'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_i 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_i. The first component is the *smoothing delay*. The worst-case delay experienced at

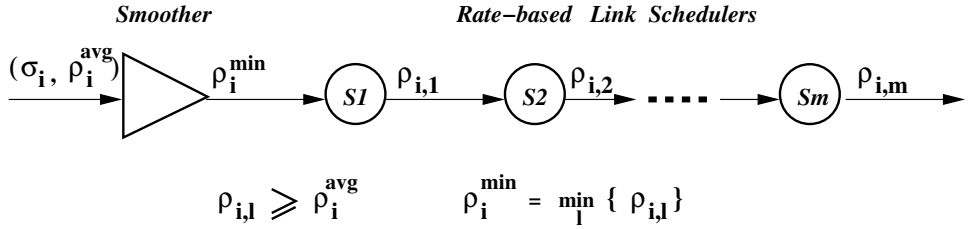

Fig. 2. Network components along a multi-hop flow path. Flow F_i that has (σ_i, ρ_i^{avg}) input traffic characterization passes through a smoother followed by a set of rate-based link schedulers. F_i's service rate at each link l is $\rho_{i,l} \geq \rho_i^{avg}$ and the bursts are smoothed before the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

the smoother by a packet from flow F_i can be shown to be as follows [5].

$$D_{i,s} = \sigma_i/\rho_i^{min} \tag{1}$$

The maximum input burst size is σ_i, the output burst size of the smoother is 0, and the output rate of the smoother is $\rho_i^{min} = \min_l\{\rho_{i,l}\}$

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) [19] [2] scheduler which is a popular approximation of the Generalized Processor Sharing (GPS) [2] class of rate-based schedulers. It can be shown [2] that the worst-case queuing delay $D_{i,l}$ experienced at link l by any packet belonging to flow F_i under WFQ service discipline is given by the following.

$$D_{i,l} = \frac{\delta_{i,l}}{\rho_{i,l}} + \frac{L_{max}}{\rho_{i,l}} + \frac{L_{max}}{C_l} \tag{2}$$

$\delta_{i,l}$ is F_i's input burst size at link l, L_{max} is the maximum packet size, $\rho_{i,l}$ is the reservation for F_i 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 $\delta_{i,l}$ 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 [20]. In general, for any rate-based scheduler, we assume that a function of the form $\mathcal{D}_l(.)$ exists that correlates the bandwidth reservation $\rho_{i,l}$ on a link l to its packet delay bound $D_{i,l}$, i.e. $D_{i,l} = \mathcal{D}_l(\rho_{i,l})$. 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) [21]) 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_i for flow F_i over an m-hop path is given by the following expression [22] [23] when each link is served by a WFQ scheduler.

$$D_i = \frac{\sigma_i}{\rho_i^{min}} + \sum_{l=1}^{m}\left(\frac{L_{max}}{\rho_{i,l}} + \frac{L_{max}}{C_l}\right) \tag{3}$$

Here $\rho_{i,l} \geq \rho_i^{avg}$ and $\rho_i^{min} = \min_l\{\rho_{i,l}\}$. 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.

V. Unicast Flow with Deterministic Delay Guarantee

We now propose a series of algorithms for delay budget partitioning that we call Load-Based Slack Sharing (LSS) 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 LSS for multicast flows; these are presented in subsequent sections. The deterministic and statistical algorithms for unicast flows are named D-LSS and S-LSS respectively and those for multicast flows are named D-MLSS and S-MLSS.

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

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

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

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

We saw in Section IV that for rate-based schedulers like WFQ, there exists a function of the form $\mathcal{D}_l(\rho_{i,l})$ that correlates a flow F_i's bandwidth reservation $\rho_{i,l}$ on a link l to its packet delay bound $D_{i,l}$. The specific form of relation $\mathcal{D}_l(\rho_{i,l})$ 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.

$\delta = 0.5$;
$b = \delta$;
while(1) {
for $l = 1$ to m {
$\rho_{N,l} = max\{(C_l - L_l - \beta_l C_l b),\ \rho_N^{avg}\}$
$D_{N,l} = \mathcal{D}_l(\rho_{N,l})$; /* Delay at link l */
}

$D_{N,s} = \sigma_N / \min_l\{\rho_{N,l}\}$; /* Smoother delay */
$slack = D_N - \sum_{l=1}^{m} D_{N,l} - D_{N,s}$;

if ($slack \geq 0$ and $slack \leq DThreshold$)
return $\hat{D_N}$;

$\delta = \delta/2$;

if ($slack > 0$)
$b = b + \delta$;
else
$b = b - \delta$;
}

Fig. 3. D-LSS: Load-based Slack Sharing algorithm for a unicast flow with deterministic delay guarantee. The algorithm returns the delay vector $\hat{D_N} =< D_{N,1}, D_{N,2}, \cdots, D_{N,m} >$.

A. Admission Control

Before computing $D_{N,l}$, one needs to determine whether the flow F_N 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_N at each link if all the residual bandwidth on the link is assigned to F_N. Thus the minimum delay budget at link l is given by $\mathcal{D}_l(C_l - L_l)$, where C_l is the total capacity and L_l is the currently reserved capacity. From Equation 1, the corresponding minimum smoothing delay is $\sigma_N / \min_l\{C_l - L_l\}$. The flow F_N can be admitted if the sum of minimum smoothing delay and per-link minimum delay budgets is smaller than D_N; otherwise F_N is rejected. More formally,

$$\frac{\sigma_N}{\min_l\{C_l - L_l\}} + \sum_{l=1}^{m} \mathcal{D}_l(C_l - L_l) \leq D_N \tag{5}$$

B. Load-based Slack Sharing (D-LSS)

Once the flow F_N 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

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

If the flow F_N can be admitted, it means that after assigning minimum delay budgets to the flow at each link, the slack ΔD_N is positive. The purpose of slack sharing algorithm (D-LSS) is to reduce the bandwidth requirement of a new flow F_N 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 sessions 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-LSS algorithm for unicast flows with deterministic delay guarantee is given in Figure 3. Let the remaining bandwidth on the l^{th} link after delay budget assignment be the form $\beta_l \times C_l \times b$. The algorithm essentially tunes the value of b in successive iterations until the resulting slack falls below a predefined *DThreshold*. Since $\beta_l \times C_l \times b$ represents the remaining capacity of the l^{th} link, β_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, β_l should be set to 1. If the optimization objective is to ensure that a link's remaining capacity is proportional to the current loads on the links, then β_l should be set to be proportional to $L_l / \sum_{i=1}^{m} L_i$ for all links l. Smaller the value of *DThreshold*, the closer LSS can get to the optimization objective.

VI. Unicast Flow with Statistical Delay Guarantee

Now we consider the case where a new flow F_N requires statistical delay guarantees (D_N, P_N) over a m-hop unicast path, i.e, the end-to-end delay of its packets needs to be smaller than D_N with a probability greater than $1 - P_N$. 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_N. The S-LSS algorithm needs to distribute D_N and P_N to constituent links of the m-hop path. In other words, it needs to assign values $D_{N,l}$ and $P_{N,l}$, such that the following partition constraints are satisfied

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

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

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 $\mathcal{D}_l(\rho_{i,l}, P_{i,l})$ and $\mathcal{P}_l(\rho_{i,l}, D_{i,l})$ that can correlate the bandwidth reservation $\rho_{i,l}$ to a statistical delay bound $(D_{i,l}, P_{i,l})$.

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

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

In Section IV, 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

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

Fig. 4. The admission control algorithm for unicast flow F_N with statistical delay requirements (D_N, P_N).

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.

A. Admission Control

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 $\{< D_{N,l}, P_{N,l} >\}$ that satisfy the partitioning constraints 7 and 8. The detailed admission control algorithm is shown in Figure 4. It starts with an initial assignment of minimum delay value $\mathcal{D}_l(C_l - L_l, 0)$ to $D_{N,l}$ assuming that all the remaining capacity of the l^{th} link is dedicated to F_N and $P_{N,l} = 0$. Since the initial assignment might violate end-to-end delay constraint, i.e. $D_{N,s} + \sum_{l=1}^{m} D_{N,l} > D_N$, the algorithm attempts to determine if there exists a looser $P_{N,l}$ such that the delay partition constraint can be satisfied. Thus the algorithm iteratively increases the value of some $P_{N,k}$ by a certain amount δ and recomputes its associated $D_{N,k}$, until either $D_{N,s} + \sum_{l=1}^{m} D_{N,l}$ becomes smaller than D_N, or $\prod_{l=1}^{m} (1 - P_{N,l})$ becomes smaller than $(1 - P_N)$. In the first case, F_N is admitted since a constraint satisfying partition exists; in the latter case even assigning all the available resources along the F_N's path is insufficient to support the QoS requested and F_N 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, δ, i.e, the one that maximizes $D_{N,k} - \mathcal{D}_k(C_k - L_k, P_{N,k} + \delta)$.

Initialize $\hat{D_N}$ and $\hat{P_N}$ with the final $D_{N,l}$ and $P_{N,l}$ values computed from the admission control algorithm;
do {
$\hat{D'_N} = \hat{D_N}$; $\hat{P'_N} = \hat{P_N}$;
$\hat{D_N}$ =relax_delay($\hat{P_N}$);
$\hat{P_N}$ =relax_prob($\hat{D_N}$);
} while((| $\hat{D_N} - \hat{D'_N}$ |$> thresh_D$) or (| $\hat{P_N} - \hat{P'_N}$ |$> thresh_P$));

Fig. 5. S-LSS: Load-based Slack Sharing algorithm for unicast flows with statistical delay guarantee.

B. Load-based Slack Sharing (S-LSS)

In the context of statistical delay guarantees, the goal of slack sharing algorithm is to apportion both the slack in delay ΔD_N and the slack in assigned probability ΔP_N over m network links traversed by the flow F_N. ΔD_N is calculated using Equation 6 and ΔP_N is calculated as follows.

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

Let the delay vector $< D_{N,1}, D_{N,2}, \cdots, D_{N,m} >$ be represented by $\hat{D_N}$. Similarly, let $\hat{P_N}$ represent the probability vector $< P_{N,1}, P_{N,2}, \cdots, P_{N,m} >$. The S-LSS algorithm for unicast flows with statistical delay guarantees is given in Figure 5. The algorithm starts with a feasible assignment of minimum delay vector $\hat{D_N}$ and probability vector $\hat{P_N}$ obtained during admission control. In every iteration, the algorithm first relaxes the delay vector $\hat{D_N}$ assuming fixed probability vector $\hat{P_N}$, and then relaxes the probability vector while fixing the delay vector. This process repeats itself till the distance between the values of $\hat{D_N}$ or between $\hat{P_N}$ from two consecutive iterations falls below a predefined threshold. Since the $\hat{P_N}$ values affect the $\hat{D_N}$ value relaxation step and vice-versa, multiple rounds of alternating relaxation steps are typically required to arrive at the final partition. The `relax_delay()` procedure is similar to the deterministic D-LSS algorithm in Figure 3 except that the resource correlation function $\mathcal{D}_l(\rho_{N,l})$ is replaced by $\mathcal{D}_l(\rho_{N,l}, P_{N,l})$. Figure 6 gives the `relax_prob()` procedure which is similar to `relax_delay()`, except that the correlation function is $\mathcal{P}_l(\rho_{N,l}, D_{N,l})$ and the slack in probability is defined as in Equation 11. In evaluations described in Section VIII, we empirically observe that this two-step iterative relaxation algorithm typically converges to a solution within 2 to 5 iterations.

C. 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

$\delta = 0.5$;
$b = \delta$;
while(1) {
    for $l = 1$ to m do {
        $\rho_{N,l} = max\{(C_l - L_l - \beta_l C_l b),\ \rho_N^{avg}\}$
        $P_{N,l} = \mathcal{P}_l(\rho_{N,l}, D_{N,l})$; /* Violation probability at link l */
    }
    $slack = \frac{\prod_{l=1}^{m}(1 - P_{N,l})}{(1 - P_N)}$;

    if($slack \geq 1$ and $slack \leq PThreshold$)
        return $\hat{P_N}$;

    $\delta = \delta/2$;

    if($slack > 1$)
        $b = b + \delta$;
    else
        $b = b - \delta$;
}

Fig. 6. relax_prob() routine to relax probability assignment vector $\hat{P_N}$, given a delay assignment vector $\hat{D_N}$.

based techniques. The approach, called Delay Distribution Measurement (DDM) based admission control, is described in detail in [8]. 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 $D_{i,l}^k$ to the worst-case delay $D_{i,l}^{wc}$, i.e., $r_k = D_{i,l}^k / D_{i,l}^{wc}$ where r_k ranges between 0 and 1. The measured samples of ratio r_k can be used to construct a cumulative distribution function (CDF) $Prob(r)$. Figure 7 shows an example of a CDF constructed in this manner in one simulation instance [8] 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 $\mathcal{D}_l(\rho_{i,l}, P_{i,l})$.

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

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

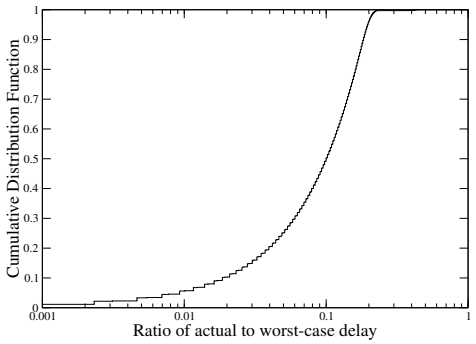

Fig. 7. Example of cumulative distribution function (CDF) of the ratio of actual delay to worst-case delay experienced by packets. X-axis is in log scale.

as follows.

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

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 [8].

VII. Partitioning for Multicast Flows

It is relatively straightforward to generalize the algorithms in Sections V and VI to multicast flows. We call these algorithms D-MLSS and S-MLSS for deterministic and statistical versions of LSS algorithm for multicast flows. In this section, we focus on the deterministic version (D-MLSS). The statistical version (S-MLSS) can be derived in a manner similar to the unicast case (S-LSS) and a detailed description of S-MLSS is provided in [8]. A real-time multicast flow F_N consists of a sender at the root and K receivers at the leaves. We assume that the end-to-end delay requirement is the same value D_N for each of the K leaves, although the number of hops from the root to each of the K leaves may be different. A multicast flow with K receivers can be logically thought of as K unicast flows, 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 paths. Here we briefly describe the essence of the admission control and delay partitioning algorithms for multicast flows and the details are presented in [8].

A. Admission Control

A K-leaf multicast flow with an end-to-end delay requirement can be admitted if and only if all of the constituent K unicast flows can be admitted. The admission control algorithm for multicast flow thus consists of applying a variant of the unicast admission control algorithm in Section V to each of the K unicast paths. The variation accounts for the fact that the K unicast paths are not completely independent. Specifically, since the K unicast paths are part of the same

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

B. Load-based Slack Sharing (D-MLSS)

To compute the final delay partition for the links of a K-leaf multicast path, we apply a variant of the D-LSS algorithm in Figure 3 to the K unicast paths one after another. Two variations deserve mention here. First the delay relaxation is applied to unicast paths $j = 1$ to K in the increasing order of their current slack in delay budget $(D_N - D_{N,s} - \sum_{l-1}^{m_j} D_{N,l})$, where $D_{N,s}$ is the smoothing delay defined in Equation 1, and m_j is the length of j^{th} unicast path.. This processing order ensures that slack partitioning along one path of the tree does not violate end-to-end delay along other paths that may have smaller slack. Secondly, when processing j^{th} unicast path, the delay relaxation only applies to those links which are not shared with paths 1 to $j-1$, i.e. those links in the j^{th} path whose $D_{N,l}$ values have not yet been determined.

VIII. Performance Evaluation

In this section, we evaluate the performance of the LSS and MLSS algorithms for unicast and multicast flows against three other schemes. The first scheme, named Equal Slack Sharing (ESS), is based on the Equal Allocation (EA) scheme proposed in [9]. EA equally partitions the end-to-end delay among the constituent links in the path of a unicast flow. As discussed in Section III, ESS 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. MESS is a multicast version of ESS in which the variations proposed in Section VII are applied. The second scheme, named Proportional Slack Sharing (PSS), is based on the Proportional Allocation (PA) scheme proposed in [7]. PA directly partitions the end-to-end QoS in proportion to loads on constituent links of unicast/multicast path. As with ESS scheme, PSS is a variant of PA that partitions the slack in end-to-end QoS in proportion to the loads on constituent links and MPSS is a multicast variant of PSS. The third scheme is the Binary-OPQ (or OPQ for short) proposed in [24] for unicast paths, which requires that the global optimization objective be expressed as the sum of per-link cost functions. We use squared sum of per-links loads, i.e. $\sum_{l=1}^{m}(L_l/C_l)^2$, as the cost to be minimized for global optimization since it captures overall loads as well as variation in loads across different links. Thus we defined the per-link cost in OPQ as $(L_l/C_l)^2$. Again, 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 [24].

Since OPQ and MOPQ operate with only a single end-to-end QoS requirement, we compare them only against the deterministic D-LSS and D-MLSS versions. On the other hand, it is straightforward to extend ESS, PSS 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-ESS and S-ESS refer to deterministic and statistical versions of ESS scheme for unicast flows, D-MESS and S-MESS refer to the same for multicast flows and so on for PSS.

The admission control algorithm that we use for ESS, PSS, OPQ, and their multicast versions is exactly the same as what we propose in this paper for LSS and MLSS. In other words, before slack sharing 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 sharing.

A. Evaluation Setup

We evaluate the performance of different slack sharing algorithms using both unicast and multicast paths. 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 mix of link capacities between 45 Mbps to 200 Mbps. Similarly, the Multicast-1 topology consists of the tree in which destinations are connected to 45 Mbps links whereas interior links have 90 Mbps capacity. The Multicast-2 topology consists of a mix of link capacities. Both unicast and multicast algorithms have also been compared over a grid topology and a general topology based on Sprint's North American IP backbone and, due to space considerations, the results are presented in [8].

All evaluations of LSS algorithms in this paper use $\beta = 1$ in the slack sharing phase since it leads to perfect load balancing with the topologies considered here. In the context of general network topologies, different values of β may be chosen depending upon the optimization objective of the routing algorithms being employed in the network. For instance, with network-wide load-balancing as the optimization criteria, results in [8] show that β values that are set in inverse proportion to expected link utilization values form a good choice for general network topologies.

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 [25], 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 $\rho_i^{avg} = 100$kbps. 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

Scenario	Topology	ESS	PSS	OPQ	LSS
Unicast/Deterministic $D_i = 60$ms	Unicast-1	219	263	271	295
	Unicast-2	225	275	284	310
Unicast/Statistical $D_i = 45$ms $P_i = 10^{-5}$	Unicast-1	81	121	N/A	319
	Unicast-2	81	117	N/A	292
Multicast/Deterministic $D_i = 60$ms	Multicast-1	221	268	265	292
	Multicast-2	219	249	244	267

TABLE I

FLOWS ADMITTED WITH ESS, PSS, OPQ, AND LSS ALGORITHMS. LENGTH=7, $\rho_i^{avg} = 100Kbps$ AND $\sigma_i = 5\ Kbits$.

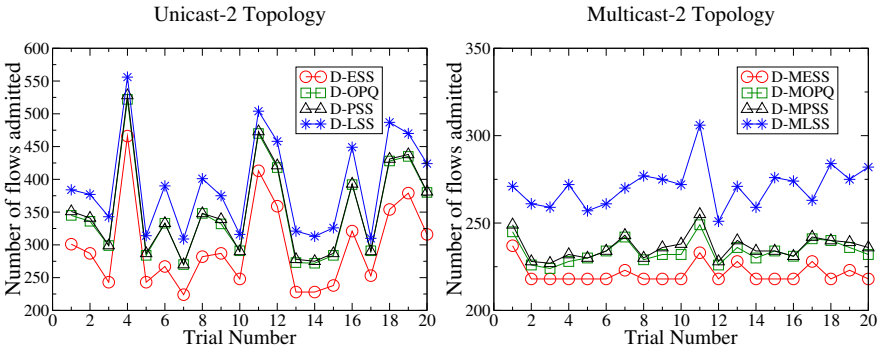

Fig. 8. Number of flows admitted with Unicast-2 and Multicast-2 topologies with deterministic delay guarantees. Hops=7, $\rho_i^{avg} = 100Kbps$, $D_i = 60ms$, $\sigma_i = 5\ Kbits$.

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. The WFQ [20] 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 cases of unicast flows with deterministic and statistical delay requirements, and multicast flows with deterministic delay requirements. For multicast flows, the memory requirements in the case of statistical trace driven simulations do not scale in our current system and hence their results are not presented.

B. Effectiveness of LSS Algorithm

We first take a snapshot view of the performance of LSS algorithm in comparison to ESS, PSS and OPQ algorithms and later examine the impact of different parameters in detail. Table I shows the number of flows admitted over a 7-hop paths for unicast and multicast flows with deterministic and statistical delay requirements. Figure 8 plots the number of flows admitted with deterministic delay guarantees under Unicast-2 and Multicast-2 topologies for different mixes of link bandwidths. The table and figures demonstrate that in all scenarios, LSS consistently admits more number of flows than all other algorithms. This is because LSS explicitly attempts to balance the loads across different links. In contrast, ESS algorithm does not optimize any specific metric and PSS 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 LSS.

The main problem lies not 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 LSS algorithm approaches the problem in a reverse fashion by exploring only those slack partitions that maintain explicit load balance among the links.

C. Capacity Evolution

In order to understand why the LSS algorithm admits a higher number of flow requests than other algorithms, we compare their resource usage patterns. Figure 9 plots the evolution of available link bandwidth on the constituent links of the Unicast-2 topology when flows require a deterministic end-to-end delay bound of 60ms. Figure 10 plots the same curves when flows require statistical end-to-end delay bound of 45ms and delay violation probability of 10^{-5}. We used a smaller statistical delay bound of 45ms compared to the deterministic delay bound of 60ms in Figure 9 because, for statistical case, the difference in performance among different algorithms is evident only at smaller delay bounds. Figure 11 plots the same curves for Multicast-2 topology with tree depth of 7 when multicast flows require end-to-end deterministic delay bound of 60ms. At any point in time, LSS is able to explicitly balance the loads on constituent links. On the other hand, the link loads are imbalanced in the case of ESS, PSS and OPQ algorithms. 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

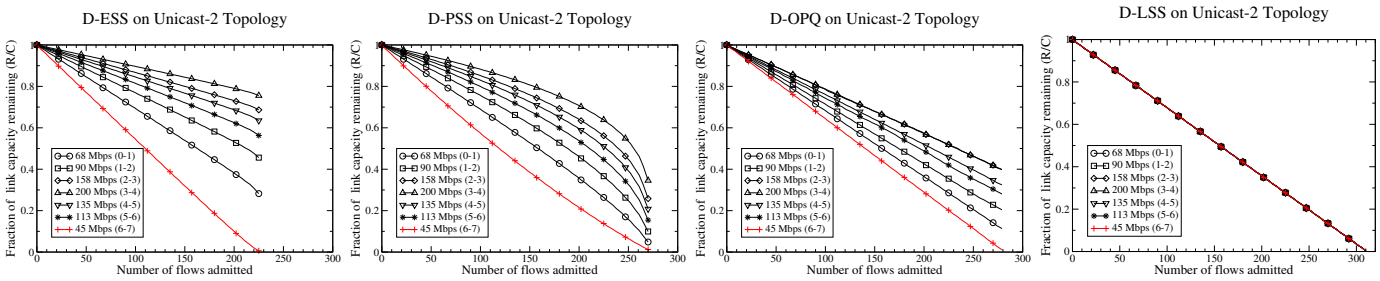

Fig. 9. Evolution of available link capacity with the number of flows admitted for unicast flows with deterministic delay guarantee. Hops=7, $\rho_i^{avg} = 100Kbps$, $D_i = 60ms$, $\sigma_i = 5\ Kbits$.

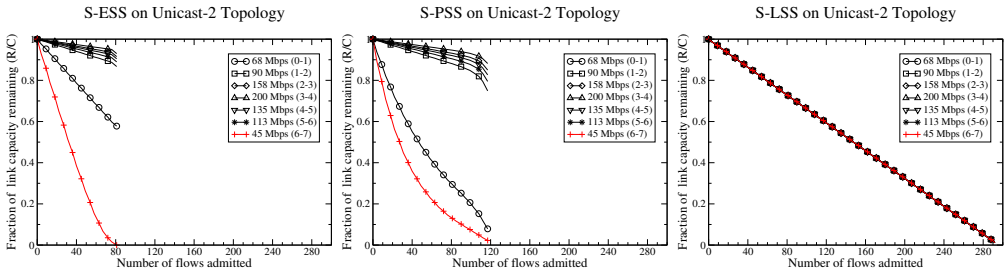

Fig. 10. Variation in available link bandwidth with the number of flows admitted for unicast flows with statistical delay guarantee. Hops=7, $\rho_i^{avg} = 100Kbps$, $D_i = 45ms$, $P_i = 10^{-5}$ and $\sigma_i = 5\ Kbits$.

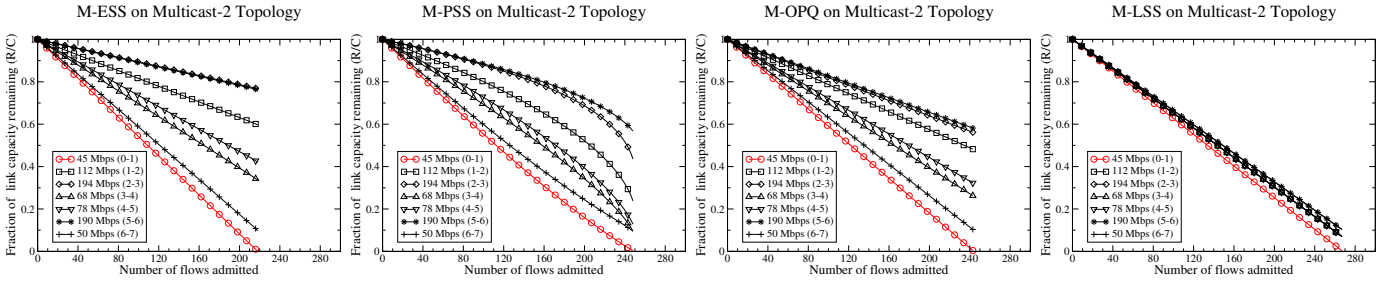

Fig. 11. Evolution of available link capacity with the number of flows admitted for multicast flows with deterministic delay guarantee. Tree depth=7, $\rho_i^{avg} = 100Kbps$, $D_i = 60ms$, $\sigma_i = 5\ Kbits$.

flows are admitted. Note that a single link with insufficient residual capacity is enough to render the entire network path unusable for newer reservations. Among ESS, PSS and OPQ algorithms, OPQ and PSS have similar performance followed by the ESS. The differences in performance arise from the extent to which each algorithm accounts for load-imbalance between links. For multicast flows in Figure 11, the capacity evolution curves are not perfectly balanced in the case of D-MLSS 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.

D. Effect of End-to-end Delay

Figure 12 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, LSS admits more flows than others since it performs load-balanced partitioning of the slack in end-to-end delay. A maximum of 450 flows with 100 Kbps average data rate can be admitted by any of the algorithms since the smallest link capacity is 45 Mbps.

E. Effect of End-to-end Delay Violation Probability

Figure 13 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 45ms end-to-end delay bound. The LSS algorithm is able to admit far more flows than ESS and PSS algorithms since it can perform load-balanced slack sharing along both the dimensions of delay as well as delay violation probability. The performance gain for LSS over ESS and PSS is much larger than in the case of deterministic delay requirements (Figure 12) because even a small increase in delay violation probability yields a significant reduction in resources assigned to a flow.

F. Effect of Path Length

Figure 14 plots the variation in number of admitted flows having deterministic delay requirements of 60ms, as the length of the unicast path increases. For all the four algorithms, there

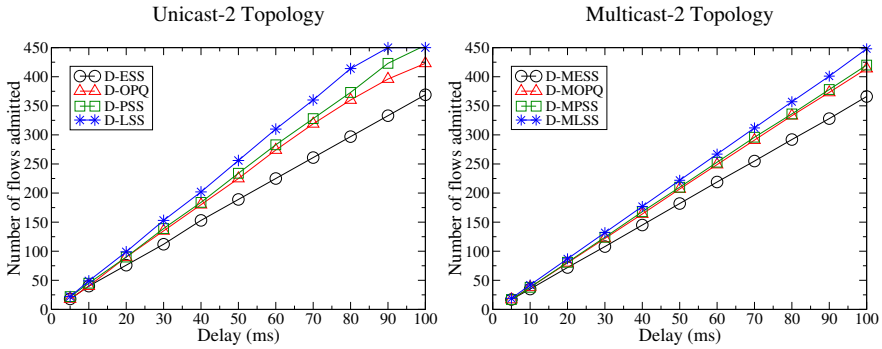

Fig. 12. Number of flows admitted vs. deterministic end-to-end delay bound for unicast and multicast paths. Hops/Tree depth=7, $\rho_i^{avg} = 100Kbps$ and $\sigma_i = 5\ Kbits$.

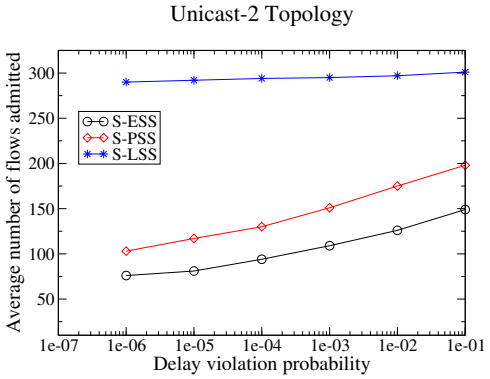

Fig. 13. Average number of flows admitted vs. end-to-end delay violation probability bound. Hops=7, $D_i = 45ms$, $\rho_i^{avg} = 100Kbps$ and $\sigma_i = 5\ Kbits$. Averages computed over ten simulation runs with different random seeds.

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. The LSS algorithm still outperforms the other three algorithms in terms of number of flows it can support since it manages the decreasing slack in delay to counter load imbalance among links.

G. Effect of Burst Size

Figure 15 plots the impact of increase in burst size on the number of flows admitted with deterministic delay requirement of $60ms$. For all the four algorithms, the number of flows admitted drops with increasing burst size. Recall that burst size contributes to the smoothing delay σ_i/ρ_i^{min} before the ingress node. For larger burst size a bigger fraction of end-to-end delay is consumed at the smoother in Figure 2 and consequently a smaller fraction of the delay is left for partition among links of the unicast/multicast path. As before, the LSS algorithm still outperforms the other three algorithms in terms of number of admitted flows.

IX. 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 load-aware delay budget partitioning algorithm that is able to 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-based Slack Sharing (LSS) algorithm allocates 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 simulation study, we have shown that the LSS algorithm can indeed admit more number of unicast or multicast flows in comparison with three other algorithms proposed in the literature. The improvement is up to 1.2 times for flows with 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 path or multicast tree, eventually these algorithms need to be integrated more closely with other components such as QoS-aware routing or inter-path load balancing to achieve global 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, Srikant Sharma and anonymous reviewers for their insightful suggestions that helped improve the presentation of this paper.

References

[1] M.S. Kodialam and T.V. Lakshman, "Minimum interference routing with applications to MPLS traffic engineering," in *Proc. of INFOCOM'2000, Tel Aviv, Israel*, March 2000, pp. 884–893.

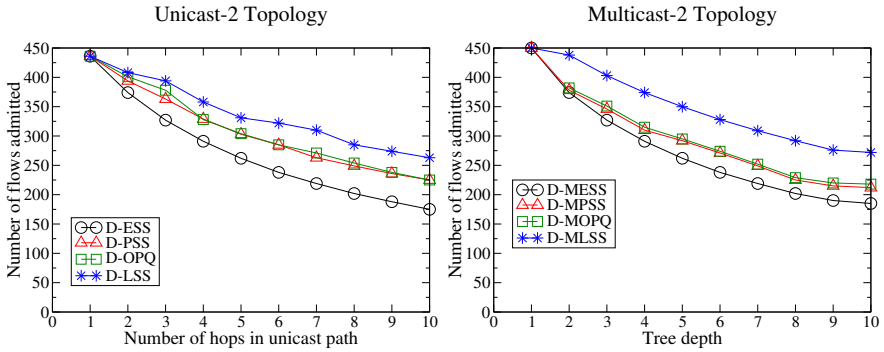

Fig. 14. Number of flows admitted vs. path length. $D_i = 60ms$, $\rho_i^{avg} = 100Kbps$ and $\sigma_i = 5\ Kbits$.

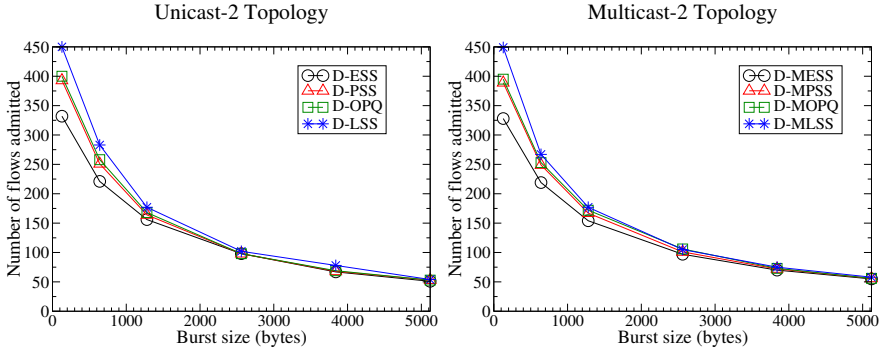

Fig. 15. Number of flows admitted vs. burst size. Hops=7, $D_i = 60ms$, and $\rho_i^{avg} = 100Kbps$.

[2] 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*, vol. 1(3), pp. 344–357, June 1993.

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

[4] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proc. of ACM SIGCOMM'90, Philadelphia, PA, USA*, Sept. 1990, pp. 19–29.

[5] 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*, vol. 4(4), pp. 482–501, August 1996.

[6] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," in *Proc. of IEEE*, Oct. 1995, vol. 83(10), pp. 1374–1396,.

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

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

[9] 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*, Jan. 1993, pp. 99–118.

[10] D.H. Lorenz and A. Orda, "Optimal partition of QoS requirements on unicast paths and multicast trees," *IEEE/ACM Transactions on Networking*, vol. 10, no. 1, pp. 102–114, Feb. 2002.

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

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

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

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

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

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

[17] 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*, May 1994, pp. 211–220.

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

[19] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," in *Proc. of ACM SIGCOMM'89*, 1989, pp. 3–12.

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

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

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

[23] 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*, vol. 2(2), pp. 137–152, April 1994.

[24] D.H. Lorenz and A. Orda, "Optimal partition of QoS requirements on unicast paths and multicast trees," in *Proc. of INFOCOM'99*, March 1999, pp. 246–253.

[25] 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.