Node Delay Assignment Strategies to Support End-to-End Delay Requirements in Heterogeneous Networks

Taieb F. Znati, *Associate Member, IEEE,* and Rami Melhem, *Fellow, IEEE*

***Abstract*—In a complex, heterogeneous network environment, such as the Internet, packets traversing different networks may be subjected to different treatments and may face different traffic loads across the routing path. This paper addresses the key issue of how to assign delay budgets to each network node along the routing path so that the end-to-end delay requirements of the supported applications are met.**

First, we describe a methodology to compute for a given flow a set of *feasible* per-node delays for the class of delay-based servers. We then formalize an optimal per-node delay assignment problem which takes into consideration the workload across the routing path. The solution, for homogeneous and heterogeneous networks, is provided. The resulting solution is *optimal*, but its implementation overhead is relatively high. To overcome this shortcoming, we propose two heuristics, *EPH()* and *LBH()*, to approximate the optimal strategy. *EPH()* uses the *equi-partition* concept to compute initial delay values and adjust these delay values to meet the end-to-end delay requirements. *LBH()* uses a *relaxation factor* to distribute the load proportionally across all nodes on the routing path. A simulation-based comparative analysis shows that the heuristics perform closely to the optimal schemes.

***Index Terms*—End-to-end delay, packet scheduling, quality of service (QoS).**

I. Introduction

WITH the ever increasing growth of the Internet, in scale and heterogeneity, the main challenge faced by current networks is to overcome the lack of service flexibility of traditional communication protocols and offer efficient solutions for quality of service (QoS) guarantees. To this end, two QoS models, namely IntServ and DiffServ, have been proposed and extensively studied in the literature. The ability to efficiently manage networks resources is critical in order to effectively address network congestion and end-to-end QoS guarantees. Networks must determine appropriate delay bounds, for each router within the networks, in order to guarantee the end-to-end delay requirements of the traffic flows. Furthermore, routers must be able to re-negotiate delays when, due to congestion or node failure along the flow's routing path, the network is no longer able to continue supporting the requested service.

In most scheduling policies used to service packets, service provisioning and traffic conditioning policies are sufficiently decoupled from the forwarding behaviors within the core network [5], [13]. The direct implication of this network design choice is the ability to implement a wide variety of service behaviors. Consequently, several scheduling schemes can be used to enforce a desired "per-hop behaviors" for different classes of services. These scheduling schemes differ in the strategies they employ to enforce rate control, the policies they use to service packets, and the types of delay guarantees they provide. In general, the mechanisms used to verify the feasibility of supporting the QoS requirements of a real-time application on a given path is based on a set of node feasibility tests. These tests integrate a packet based workload characterization into the feasibility equation of the scheduling discipline used to verify the possibility of supporting the delay requirement by each node on the routing path such that the end-to-end delay requirement of the application is met. Very few of these schemes, however, describe the mechanisms used to assign either the per-node delay or the service rate required across the routing path in order to meet the end-to-end delay requirements.

The main objective of this paper is to address the question of how to efficiently assign per-node delays to a new session, or equivalently how to assign ϕ's values to a general processor sharing (GPS) session along the routing path, so that the end-to-end delay requirement of the new session are met without violating the delay requirements of currently supported sessions. It is worth noting that the paper does not propose a new packet scheduling algorithm, nor does it seek to introduce a new calculus to derive delay bounds for a given packet service discipline. Rather, the paper addresses the delay budget assignment problem first described in [18], and proposes a framework within which an optimal, load-based per-node delay strategy can be developed. To this end, we first propose a methodology to compute a range of *feasible* delay budgets for each node on the routing path, given the flow's end-to-end delay requirement and the resources currently available at the routing node. Based on the computed range of feasible delays, an optimal, load-based strategy, along with heuristics on how to assign per-node delays across the routing path, are described. Both homogeneous as well as heterogeneous networks are considered.

The rest of the paper is organized as follows. In Section II, we review the related work. In Section III, a methodology for computing per-node delays is presented. In Section IV, three per-node delay assignment strategies are described. In Section V,

Manuscript received September 15, 2000; revised June 21, 2002; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Knightly. This work was supported in part under a National Science Foundation (NSF) award MIP 96-33729 to the University of Pittsburgh.

The authors are with the Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 USA (e-mail: znati@cs.pitt.edu; melhem@cs.pitt.edu).

Digital Object Identifier 10.1109/TNET.2004.836141

the performances of the strategies are discussed and compared for different types of network environments. The last section provides the conclusion of this work.

II. Related Work

Several scheduling schemes have been proposed to service traffic and achieve a desired level of QoS Support [3], [10], [22], [25]. Some of these schemes seek to maintain a high level of fairness while reducing the complexity of emulating the GPS policy [11], [20]. A class of these schemes seeks to simplify the scheduling algorithm by employing improved data structures for storing the set of flow priorities [21], [24]. A second class of these schemes attempts to reduce the complexity of their emulation by improving upon the virtual time computation used to simulate the ideal server's behavior [2]. A third class of scheduling disciplines, referred to as fair airport scheduling algorithms, maintains an auxiliary and a guaranteed service queue to dynamically control the level of fairness in order to provide a flow with differing worst-case delay bounds given without exceeding the delay bounds by more than what the application can tolerate [4], [8].

The above schemes are based on the idealized GPS scheduling policy and, in most cases, seek to generalize this policy based on some form of service definition. Very few of these schemes, however, address the question of how to assign the proper weight to a new session in a way such that, after accepting the new session, every session currently supported by the server is guaranteed its average rate and worst-case end-to-end delay. An approach, proposed by Parekh [18], computes the largest and smallest values of ϕ that would ensure the worst case delay for the new session and selects the midpoint of the interval resulting from these extreme points. It is clear, however, that this strategy may not be efficient.

A class of servers based on the concept of universal curves, or a derivative thereof, have been introduced [9], [19]. These schemes seek to support large schedulability regions, given a set of delay-bound requirements and traffic burstiness specifications. In this case also the proposed schemes do not specifically address issues related to per-node delay assignment across a routing path.

A closely related work to the work presented in this paper provides algorithms for flow admission control in the context of a rate-controlled model based on the earliest deadline first (EDF) scheduling policy [7]. The framework assumes a homogeneous set of EDF-based servers and derives the minimum delay that can be guaranteed by a flow. Here again, the scheme does not address the issue of how to assign efficiently per-node delays along a routing path so that a network defined performance metric is enforced.

III. Per-Node Delay Computation

Based on the traffic specification and the current excess buffering and processing capacities of a node, the feasibility of supporting a new flow, without violating the delay bounds of currently supported flows, can be assessed. In the following, we first present the flow workload model and the processing and buffer capacity model. We then proceed to describe the methodology used to compute feasible delays at a given node. It is worth noting that the methodology does not seek to produce exact delay bounds for specific scheduling policies, as described in [14] and [23]. Rather, the framework seeks to determine a range of feasible per-node delay budgets which can then be used to assign per-node delays in a way such that the load remains balanced across the routing path.

A. Flow Workload Model

The capability of a node to produce a range of feasible delay values for a given network flow requires a mechanism by which a node can determine the amount of service time required by a network flow over a given time interval. Consider a routing node, i, and a real-time flow, j, characterized by its maximum end-to-end delay value, Δ_j, and its Linear Bounded Arrival Process traffic rate specification vector, (b_j, r_j, n_j), where n_j is the number of packets generated over r_j and b_j is the maximum packet burst size over any time interval, so that the application's long-term packet rate is (n_j/r_j).

The maximum number of packets, $\sigma_j(t)$, generated by j over a time interval of size t can be expressed as

$$\sigma_j(t) = b_j + \left\lceil \frac{t}{r_j} \right\rceil \cdot n_j. \tag{1}$$

Let $\mu_{j,i} = (P_j/R_i)$ represent the maximum amount of service time required to process a packet from flow j at node i, where P_j is the packet size of flow j, and R_i is the service rate at node i. Furthermore, given the traffic rate specified by j, the maximum amount of service time required by j over an interval of size t can be expressed as $\mathcal{W}_{j,i}(t) = \sigma_j(t) \cdot \mu_{j,i}$.

In the above, js workload is specified by computing the maximum number of intervals of size r_j that can fit in t and multiplying this value by the number of packets that can be generated per r_j interval. This load characterization provides an *upper bound* on the amount of traffic generated by flow j over a time interval t.

B. Resource Capacity Models

To accommodate a rich mixture of network services and applications and achieve efficient use of the network resources, different packet scheduling policies are likely to be widely deployed. In this paper, we consider three classes of scheduling policies, namely deadline-based, delay-based and first-in-first-out.

The deadline-based scheduling policy dynamically assigns priorities to packets based on their deadlines. The packet with the closest deadline is assigned the highest priority. This policy has been shown to have a maximum schedulability region for a given set of delay vectors [12]. The delay-based scheme assigns priorities to flows so that flows with shorter delays have higher priorities than those with longer delays. It has been shown that this policy is optimal among fixed-priority scheduling algorithms [15]. The first-in-first-out policy, on the other hand, is simple to implement and widely used among Internet routers. Its capability to support adequate delay bounds, however, is limited.

A closer look at the delay bound characteristics supported by delay-based network servers reveals that the minimum and maximum feasible delays are correlated with the amount of processing and buffer capacities available at intermediate nodes [6]. Smaller per-node delays reduce the effective buffer requirements of the network flow but increase its effective processing requirements. Larger per-node delays, however, result in increased buffer holding times, which in turn increases the effective buffer requirements of the flow. Based on the above observation, the characterization of the *smallest feasible delay bound* $l_{j,i}$, of flow j at node i for a fixed amount of buffers, becomes a factor of node is *excess processing capacity*. Consequently, given a sufficient amount of buffers to hold flow j's traffic, $l_{j,i}$ can be exclusively determined based on the amount of processing capacity that can be allocated to j without violating the traffic requirements of the remaining flows currently supported by node i.

On the other hand, the characterization of the *largest feasible delay bound* $u_{j,i}$ of flow j at node i, is directly correlated to the buffering capacity of the node. Assuming that a sufficient amount of processing capacity is available, the largest delay value $u_{j,i}$ that can be supported by node i is exclusively dependent on the amount of buffering capacity that can be allocated to flow j without violating the traffic requirements of the currently supported flows. Consequently, the characterization of $u_{j,i}$ can be achieved based exclusively on node i's *excess buffer capacity*.

In the following, we present models to characterize the processing and buffer capacity of a network server. We then show how these models can be used to derive the smallest and largest feasible delay bounds, $l_{j,i}$ and $u_{j,i}$, respectively.

1) Processing Capacity Service Model: For a given class of delay-based servers, one can derive a collection of practical and theoretically sound quantitative methods which can be used by the server to assess the feasibility of supporting the real-time requirements of a new flow while continuing to guarantee the requirements of currently supported flows. This can be achieved by computing the utilization of the scheduled activities and comparing it to a schedulable bound which depends on the scheduling policy used by the server.

Let i be a network node which uses a nonpreemptive deadline-based scheduling policy to service packets from different flows, and consider a set of flows, $j = 1, 2, \ldots N$, traveling through i. Each flow j is characterized by its traffic specification vector, (b_j, r_j, n_j) and its end-to-end delay Δ_j. Furthermore, packets from flow j encounter a delay, $\Delta_{j,i}$ at node i. The support of flows $j = 1, 2, \ldots N$ are feasible at node i if

$$\sum_{j=1}^{N} \frac{\mathcal{W}_{j,i}(\Delta_{j,i})}{\Delta_{j,i}} \leq 1 - \frac{\max_{1\leq j\leq N}\{\mu_{j,i}\}}{\min_{1\leq j\leq N}\{\Delta_{j,i}\}} \tag{2}$$

where $\mathcal{W}_{j,i}(\Delta_{j,i})$ is the maximum amount of service time required by flow j over its assigned delay, $\Delta_{j,i}$, at node i. The term $(\max_{1\leq j\leq N}\{\mu_{j,i}\}/\min_{1\leq j\leq N}\{\Delta_{j,i}\})$ accounts for the nonpreemptive aspect of the packet service at node i. It represents the maximum amount of time a higher priority packet, arriving just at the instant a lower priority packet gained access to the server, may be forced to wait before being serviced.

For a nonpreemptive delay-based scheduling policy, the feasibility test to support a set of flows, $j = 1, 2, \ldots, N$, at node i can be expressed as

$$\sum_{j=1}^{N} \frac{\mathcal{W}_{j,i}(\Delta_{j,i})}{\Delta_{j,i}} \leq N \cdot (2^{(1/N)} - 1) - \frac{\max_{1\leq j\leq N}\{\mu_{j,i}\}}{\min_{1\leq j\leq N}\{\Delta_{j,i}\}}. \tag{3}$$

Note that, the first-in-first-out scheduling policy can be viewed as a special case where the server can only support a constant delay value for all flows.

For a given routing node i and a set of N flows, with delays, $\Delta_{1,i}, \ldots, \Delta_{j,i}, \ldots, \Delta_{N,i}$, and processing requirements, $\mathcal{W}_{1,i}(\Delta_{1,i}), \ldots \mathcal{W}_{j,i}(\Delta_{j,i}), \ldots, \mathcal{W}_{N,i}(\Delta_{N,i})$, respectively, the feasibility test for the above nonpreemptive, delay-based servers can be expressed in a general form as

$$\sum_{j=1}^{N} \frac{\mathcal{W}_{j,i}(\Delta_{j,i})}{\Delta_{j,i}} \leq P_i - \frac{\bar{\mu}}{\underline{\Delta}} \tag{4}$$

where $\bar{\mu} = \max_{1\leq j\leq N}\{\mu_{j,i}\}$ and $\underline{\Delta} = \min_{1\leq j\leq N}\{\Delta_{j,i}\}$.

The term P_i denotes the total percentage of service capacity which can be allocated to provide guaranteed service to real-time flows at node i, and $(\max_{1\leq j\leq N}\{mu_{j,i}\}/\min_{1\leq j\leq N}\{\Delta_{j,i}\})$ represents the amount of service rate required to account for the nonpreemptive nature of the server at node i. For a deadline-based scheduler, the value of P_i equals 1, while for a delay-based scheduler P_i is $N \cdot (2^{(1/N)} - 1)$.

An amount P_i^a of the total capacity, P_i, is allocated to support the real-time requirements of the currently supported flows at node i, while an amount P_i^v is reserved to account for potential nonpreemptiveness. The excess capacity, P_i^x, is the percentage of node i's total processing capacity which can be allocated to support new network flows. At any instant, P_i^x satisfies $P_i^x = P_i - P_i^a - P_i^v$.

2) Buffer Capacity Model: In order to accommodate different types of servers, the buffer capacity model proposed in this paper considers both work-conserving and nonwork-conserving servers. A work-conserving server is idle only when no packets are awaiting servers, while a nonwork-conserving server may suspend service even if the traffic queues are not empty.

In this model, each node is configured with memory to buffer packets as they are queued for service. The *total buffering capacity* B_i determines the total number of packets which can be queued either in the waiting rooms or service queues at node i. The waiting room is used to absorb the jitter introduced upon a flow's packets at the previous upstream node. A packet is considered *early* at a given node if the actual time spent at the upstream node is less than the original delay assigned to the packet. The *earliness* of a packet determines the amount of time a packet is held in the waiting room before it is considered eligible for service and moved to its appropriate service queue. By absorbing the jitter, the traffic pattern of a flow can effectively be reconstructed at each switching node to the form it had when it entered the network.

At any time, an amount B_i^a of node i's total capacity B_i is allocated to accept flows such that no packets are dropped due to lack of buffer space. Furthermore, at any time, B_i^x denotes the number of excess buffers which can be allocated to handle packets from future network flows. Consequently, $B_i^x = B_i - B_i^a$ holds.

The processing and buffer capacity models described above can be used by a given routing node i to derive the smallest and largest feasible delay bounds, $l_{j,i}$ and $u_{j,i}$, for a flow j, characterized by its rate specification vector, (b_j, r_j, n_j). These delay bounds provide a range of feasible delays that can be supported by the node i.

C. Smallest Feasible Delay Value

The characterization of the processing excess capacity is the basis for the computation of the smallest per-node delay bound that can be supported by a node. More specifically, let $\mathcal{W}_{N+1,i}(t)$ represent the maximum workload required by a new flow $N+1$ at node i over a potential delay bound t. As described above, the exact criterion for the new flow to be accepted by a delay-based server at node i, without violating the requirements of the flows currently supported by i, can be expressed as

$$\sum_{j=1}^{N} \frac{\mathcal{W}_{j,i}(\Delta_{j,i})}{\Delta_{j,i}} + \frac{\mathcal{W}_{N+1,i}(t)}{t} \leq P_i - \frac{\bar{\mu}}{\min(\underline{\Delta}, t)}.$$

Substituting $\mathcal{W}_{N+1,i}(t)$ by its packet workload-based value, results in

$$\frac{b_{N+1} + \left\lceil \frac{t}{r_{N+1}} \right\rceil \cdot n_{N+1}}{t} \leq \underbrace{P_i - \sum_{j=1}^{N} \frac{\mathcal{W}_{j,i}(\Delta_{j,i})}{\Delta_{j,i}}}_{P_i^x} - \underbrace{\frac{\bar{\mu}}{\min(\underline{\Delta}, t)}}_{P_k^v}.$$

The term P_i^x in the above equation denotes node i excess processing capacity, while $(b_j + \lceil (t/r_j) \rceil \cdot n_j/t)$ represents the processing requirements of flow j over a delay t. The value $t = l_{N+1,i}$, for which the equality holds, specifies a lower bound on the delay values node i can offer to $N+1$, based on i's current excess capacity and $N+1$'s requirements. This value is achieved by dedicating all node i's excess processing capacity to flow $N+1$.

Noting that if $b_{N+1} + ((t/r_{N+1}) + 1) \cdot n_{N+1}$ packets can be processed on time by the server, then $b_{N+1} + \lceil (t/r_{N+1}) \rceil \cdot n_{N+1}$ can also be processed on time by the server, the smallest feasible delay, $l_{N+1,i}$ can thus be expressed as

$$l_{N+1,i} = \begin{cases} \dfrac{b_{N+1} + n_{N+1} + \bar{\mu}}{P_i^x - \dfrac{n_{N+1}}{r_{N+1}}}, & \text{if } t \leq \underline{\Delta} \\ \dfrac{b_{N+1} + n_{N+1} + \bar{\mu}}{P_i^x - \dfrac{1}{\underline{\Delta}} - \dfrac{n_{N+1}}{r_{N+1}}}, & \text{otherwise.} \end{cases} \quad (5)$$

D. Largest Feasible Delay Value

The upper bound delay, $u_{N+1,i}$, represents the maximum *total* delay a packet from flow $N+1$ can be delayed at node i without causing flow $N+1$ to loose any of its packets and without violating the buffer requirements of the remaining flows which are currently supported by node i. This delay encompasses the total time a packet from flow $N+1$ may wait in i's waiting rooms and service queues, and can be computed based on the current excess buffer capacity of node i. Taking into consideration the requirements of currently supported flows, the buffer capacity constraint at node i can be expressed as

$$B_{1,i} + \cdots + B_{N,i} + B_{N+1,i} \leq B_i \quad (6)$$

where $B_{j,i}$ denotes the number of buffers allocated to flows $j = 1, \ldots, N+1$ to guarantee that none of their packets are dropped. Since packets from $N+1$ may be queued in their associated waiting room for up to $\Delta_{N+1,i-1}$ units of time and in the packet service queue for no more than $\Delta_{N+1,i}$ units of time, the maximum amount of time a packet from $N+1$ can be queued at i is $(\Delta_{N+1,i-1} + \Delta_{N+1,i})$. Consequently, the number of buffers, $B_{N+1,i}$, required to ensure a loss-free service to $N+1$ must be sufficient to hold the maximum number of packets generated by $N+1$ over $(\Delta_{N+1,i-1} + \Delta_{N+1,i})$. This number can be expressed as

$$B_{N+1,i} = b_{N+1} + \left\lceil \frac{\Delta_{N+1,i-1} + \Delta_{N+1,i}}{r_{N+1}} \right\rceil n_{N+1}. \quad (7)$$

Combining (6) and (7) and isolating the $B_{N+1,i}$ term results in

$$b_{N+1} + \left\lceil \frac{\Delta_{N+1,i-1} + \Delta_{N+1,i}}{r_{N+1}} \right\rceil n_{N+1} \leq \underbrace{B_i - \sum_{j=1}^{N} B_{j,i}}_{B_i^x}.$$

The above equation holds if

$$b_{N+1} + \left(\frac{\Delta_{N+1,i-1} + \Delta_{N+1,i}}{r_{N+1}} + 1 \right) n_{N+1} \leq B_i^x.$$

Solving for $\Delta_{N+1,i-1} + \Delta_{N+1,i}$ results in

$$\Delta_{N+1,i-1} + \Delta_{N+1,i} \leq \frac{r_{N+1}}{n_{N+1}} \left[B_i^x - b_{N+1} - n_{N+1} \right].$$

Let $u_{N+1,i}$ specify an upper bound on delay values node i can offer to $N+1$, based on i's current buffer capacity and $N+1$'s requirements. This value is obtained by equating the amount of buffers, $b_{N+1} + \lceil (\Delta_{N+1,i-1} + \Delta_{N+1,i}/r_{N+1}) \rceil n_{N+1}$, necessary to satisfy the requirements of $N+1$ over a delay $(\Delta_{N+1,i-1} + \Delta_{N+1,i})$, to the excess buffer capacity, B_i^x, at node i. Therefore, if $(B_i^x - b_{N+1} - n_{N+1}) > 0$, a feasible value for $u_{N+1,i}$ can be expressed as follows:

$$u_{N+1,i} = \frac{r_{N+1}}{n_{N+1}} \left[B_i^x - b_{N+1} - n_{N+1} \right]. \quad (8)$$

IV. Delay Assignment Strategies

Given a request to establish a new flow $N+1$ characterized by its traffic rate specification vector $(b_{N+1}, r_{N+1}, n_{N+1})$ and its maximum end-to-end delay value Δ the methodology presented above leads to a range of feasible delay values, $[l_{N+1,i}, u_{N+1,i}]$, that can be supported by a given routing node i. The objective is to assign, from within the range of feasible delays, a per-node delay value $\delta_{N+1,i}$ for each node i along the routing path, such that the maximum end-to-end delay of flow $N+1$ is not exceeded. It is clear, that assigning the smallest feasible delay value at a node along the routing path is not advisable as it exhausts the entire processing capacity of the node, thereby causing the rejection of any subsequent flows. The same observation can be made with respect to assigning the feasible largest delay value to the flow, since such an assignment is likely to exhaust the buffer capacity of the routing node. Sections IV-A–D, we propose and discuss more efficient strategies for flexible delay assignment along a flow's routing path.[1]

A. Optimal Load-Based Strategy

Requests for flow establishment arrive to the network and are processed sequentially. Assume the network receives a request to establish a new flow characterized by its rate specification vector (b, r, n) and its maximum end-to-end delay value Δ. Furthermore, assume that the routing path is composed of nodes $1, \ldots, K$. A load-based, per-node delay assignment strategy is said to be optimal if: 1) the per-node delay assignments are feasible across the routing path; 2) the end-to-end delay requirements of the new flow are enforced without violating the delay requirements of the existing flows; and 3) the load at each node along the routing path is balanced. This assignment will guarantee that, for the same workload, heavily loaded nodes will be assigned larger per-node delays than lightly loaded nodes, thereby reducing the likelihood of bottlenecks across a routing path.

The "optimal" delay assignment problem can be formalized as follows. **Find** $\delta_1, \ldots, \delta_K$ **such that** $\sum_{i=1}^{K} \rho_i + (\mathcal{W}(\delta_i)/\delta_i)$ **is minimum, subject to** $\sum_{i=1}^{K} \delta_i \leq \Delta$ and $l_i \leq \delta_i \leq u_i$ for $1 \leq i \leq K$. In this formulation, ρ_i is the current workload at node i just before the new flow request arrives, and $\mathcal{W}(\delta_i)$ is the workload requested by the new flow over the time period δ_i.

Noting that the maximum workload that a node i can observe over an interval of time δ_i is $b + ((\delta_i/r) + 1) \cdot n$, the objective function of the delay assignment problem becomes $\sum_{i=1}^{K} \rho_i + (n\mu_i/r) + \sum_{i=1}^{K} ((b+n)\mu_i/\delta_i)$. Furthermore, since $\sum_{i=1}^{K} \rho_i + (n\mu_i/r)$ is independent of δ_i, the optimization problem simplifies to: **Find** $\delta_1, \ldots, \delta_K$ **such that** $\sum_{i=1}^{K} (\mu_i/\delta_i)$ **is minimum, subject to:** $\sum_{i=1}^{K} \delta_i \leq \Delta$ and $l_i \leq \delta_i \leq u_i$ for $1 \leq i \leq K$.

In the following, we first solve the optimization problem in the case of homogeneous networks, where the service rate μ_i for all nodes $1 \leq i \leq K$ across the routing path, is the same. We then describe the more elaborate solution for the case of a heterogeneous network of servers.

B. Case of Homogeneous Networks

In the case of a homogeneous network, since $\mu_i = \mu$, $i = 1, \ldots, K$, the optimal per-node delay assignment problem reduces to

Find $\delta_1, \ldots, \delta_K$ **such that** $\sum_{i=1}^{K} (1/\delta_i)$ **is minimum, subject to**

$$\sum_{i=1}^{K} \delta_i \leq \Delta, \quad \text{and} \quad l_i \leq \delta_i \leq u_i \tag{9}$$

where Δ is the maximum end-to-end delay value of the new flow, and l_i and u_i represent the lower and upper bound per-node delay values at node i, respectively. It is clear that if $\sum_{i=1}^{K} l_i \leq \Delta$ does not hold true then no solution exists to the above problem and the flow must be rejected. A feasible solution to this problem is the one that satisfies both constraints of (9) without necessarily minimizing the objective function. A solution is optimal if it minimizes the objective function in addition to satisfying the constraints of (9). To arrive at the optimal solution, we initially disregard the objective function and consider the set of possible feasible solutions. This set can then be used to determine the optimal solution.

[1]For notational simplicity, the index $N+1$ denoting the flow is dropped when no ambiguity ensues.

The optimal solution for the unbounded problem, which does not necessarily satisfy the constraints of (9), can be obtained using partial differentiation and solving the resulting linear equations in K variables. The solution can be shown to be the one in which the delays δ_is are all equal to $\xi = \Delta/K$, a varying value which we refer to in this paper as the *equi-partitioned value*. It is worth noting that this solution is also optimal for the bounded problem in the trivial case when the *equi-partitioned value* lies between all the l_is and u_is. It is also clear that any member of a set of feasible solutions is a solution in which all the K nodes on the routing path of the new flow belong to one of three disjoint sets, LF, UF, and NF, depending on whether the node assigns to the new request the smallest delay value, l_i, the largest delay value, u_i, or a value in between the smallest and largest delay values. These observations provide the basic intuition to arrive at the optimal solution. Prior to deriving the optimal solution, we prove a theorem that states the conditions which must be met in order for the solution to be optimal.

Theorem 1: Optimality Theorem: Consider a solution, $\vec{\delta_o} = (\delta_1, \ldots, \delta_K)$, which satisfies the constraints of (9) by partitioning the nodes $\{1, \ldots, K\}$ into three disjoint sets LF, UF, and NF, of sizes n, m and $n_f = K - n - m$, respectively, and assigning values to $\vec{\delta}$ as follows:

- $\forall$ nodes $1 \leq i \leq K \in LF$, $\delta_i = l_i$;
- $\forall$ nodes $1 \leq i \leq K \in UF$, $\delta_i = u_i$;
- $\forall$ nodes $1 \leq i \leq K \in NF$, $\delta_i = \xi = \left(\Delta - \sum_{i \in LF} l_i - \sum_{i \in UF} u_i\right)/n_f$.

The above solution minimizes the objective function if

- $\forall$ nodes $1 \leq i \leq K \in LF$, $\xi \leq l_i$;
- $\forall$ nodes $1 \leq i \leq K \in UF$, $u_i \leq \xi$;
- $\forall$ nodes $1 \leq i \leq K \in NF$, $l_i < \xi < u_i$.

Proof: The proof follows from Kuhn–Tucker necessary and sufficient conditions for a global optimal solution to a problem with inequality and equality constraints [16]. We first rewrite the optimization problem in a form more suited to the application of Kuhn–Tucker conditions

$$\begin{aligned} \textbf{Minimize} \quad & f(\vec{\delta}) = \sum_{i=1}^{K} \frac{1}{\delta_i} \\ \textbf{Subject to:} \quad & g_i(\vec{\delta}) = l_i - \delta_i \leq 0 \text{ for } i = 1, \ldots, K \\ & g_{K+i}(\vec{\delta}) = \delta_i - u_i \leq 0 \text{ for } i = 1, \ldots, K \\ & h(\vec{\delta}) = \sum_{i=1}^{K} \delta_i - \Delta = 0. \end{aligned}$$

Given that the functions f, g_i, $i = 1, \ldots, 2 \cdot K$, and h are all convex, the Kuhn–Tucker's optimality conditions state that a solution $\vec{\delta_o}$ to the above problem is globally optimal *if and only if* there exist a scalar $\lambda_j \geq 0$ for every $j \in I = \{j : g_j(\vec{\delta_o}) = 0\}$, and a scaler v such that [16]

$$\nabla f(\vec{\delta_o}) + \sum_{j \in I} \lambda_j \nabla g_j(\vec{\delta_o}) + v \nabla h(\vec{\delta_o}) = 0. \quad (10)$$

We will prove that the solution $\vec{\delta_o}$ given by Theorem 1 satisfies the above conditions, and thus is optimal. Without loss of generality, assume that $LF = \{1, \ldots, n\}, UF = \{n+1, \ldots, n+m\}$ and $NF = \{n+m+1, \ldots, K\}$. Hence, the *equi-partitioned value*, $\xi = \left(\delta - \sum_{i=1}^{n} l_i - \sum_{i=n+1}^{n+m} u_i\right) / (K-n-m)$. For $f(\vec{\delta}) = \sum_{i=1}^{K} (1/\delta_i)$, we have

$$\nabla f(\vec{\delta}) = \left[\frac{-1}{\delta_1^2} \quad \frac{-1}{\delta_2^2} \quad \cdots \quad \frac{-1}{\delta_K^2} \right]^T.$$

Based on the above, we have

$$\nabla f(\vec{\delta_o}) = \left[\frac{-1}{l_1{}^2} \cdots \frac{-1}{(l_n)^2} \quad \frac{-1}{(u_{n+1})^2} \cdots \frac{-1}{(u_{n+m})^2} \quad \frac{-1}{\xi^2} \cdots \frac{-1}{\xi^2} \right]^T.$$

From the definition of $g()$, we have

$$\nabla g_j(\vec{\delta}) = \left[0 \ldots 0 \underbrace{-1}_{jth} 0 \ldots 0 \right]^T$$

$$\nabla g_{K+j}(\vec{\delta}) = \left[0 \ldots 0 \underbrace{1}_{jth} 0 \ldots 0 \right]^T.$$

Moreover, since $g_k(\vec{\delta_o}) = 0$ for $k = 1, \ldots, n$ and for $k = K+n+1, \ldots, K+n+m$, hence $I = \{1, \ldots, n, K+n+1, \ldots, K+n+m\}$. This leads to (11).

We also have $v \, \nabla h(\vec{\delta_o}) = v \cdot [1 \ldots 1]^T = [v \ldots v]^T$. Therefore, (10) can be rewritten as

$$\begin{bmatrix} \frac{-1}{(l_1)^2} \\ \vdots \\ \frac{-1}{(l_n)^2} \\ \frac{-1}{(u_{n+1})^2} \\ \vdots \\ \frac{-1}{(u_{n+m})^2} \\ \frac{-1}{\xi^2} \\ \vdots \\ \frac{-1}{\xi^2} \end{bmatrix} + \begin{bmatrix} -\lambda_1 \\ \vdots \\ -\lambda_n \\ \lambda_{K+n+1} \\ \vdots \\ \lambda_{K+n+m} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} v \\ \vdots \\ v \\ v \\ \vdots \\ v \\ v \\ \vdots \\ v \end{bmatrix} = 0.$$

Using the above, we derive the following:

$$\frac{-1}{l_i{}^2} - \lambda_i + v = 0 \text{ for } i = 1, \ldots, n \quad (12)$$

$$\frac{-1}{u_i{}^2} + \lambda_{K+i} + v = 0 \text{ for } i = n+1, \ldots, n+m \quad (13)$$

$$\frac{-1}{\xi^2} + v = 0 \text{ for } i = n+m+1, \ldots, K. \quad (14)$$

```
Algorithm Opt()
1. $k = 0$ ; $n_f = K$ ; $LF_0 = UF_0 = \emptyset$ ; $\xi_0 = \Delta/n_f$.
2. While $\exists i$ such that $i \notin UF_k$ and $u_i \leq \xi_k$ do /* add $i$
   to $UF$ */
    • $UF_{k+1} = UF_k \cup \{i\}$ ; $LF_{k+1} = LF_k$
    • $\xi_{k+1} = (\xi_k \; n_f - u_i)/(n_f - 1)$
    • $n_f = n_f - 1$
    • $k = k + 1$.
3. While $\exists j$ such that $j \notin LF_k$ and $l_j \geq \xi_k$ do /* add $j$
   to $LF$ */
    • $LF_{k+1} = LF_k \cup \{j\}$ ; $UF_{k+1} = UF_k$
    • $\xi_{k+1} = (\xi_k \; n_f - l_i)/(n_f - 1)$
    • $n_f = n_f - 1$; $k = k + 1$
    • Repeat /* check if any node should be removed
      from $UF$ */
        * find $i$ such that $u_i = max_{q \in UF_k}\{u_q\}$
        * if $(u_i > \xi_k)$ { /* remove $i$ from $UF$ */
            - $UF_{k+1} = UF_k - \{i\}$ ; $LF_{k+1} = LF_k$
            - $\xi_{k+1} = (\xi_k \; n_f + u_i)/(n_f + 1)$
            - $n_f = n_f + 1$; $k = k + 1$. }
      Until $u_i \leq \xi_k$
```

Fig. 1. Optimal delay assignment.

From (14), we obtain $v = 1/\xi^2$. Thus, for $i = 1, \ldots, n$, (12) implies that $\lambda_i = v - 1/l_i{}^2 = 1/\xi^2 - 1/l_i{}^2 = (l_i{}^2 - \xi^2)/(l_i{}^2 \cdot \xi^2)$. Given that the solution $\vec{\delta_o}$ satisfies $l_i \geq \xi$ for $i \in LF$, we conclude that $\lambda_i \geq 0$ for $i = 1, \ldots, n$.

Similarly, for $i = n+1, \ldots, n+m$, (13) implies that $\lambda_{K+i} = -v + 1/u_i{}^2 = -1/\xi^2 + 1/u_i{}^2 = (-u_i{}^2 + \xi^2)/(u_i{}^2 \cdot \xi^2)$. Given that the solution $\vec{\delta_o}$ satisfies $u_i \leq \xi$ for $i \in UF$, we conclude that $\lambda_{K+i} \geq 0$ for $i = n, \ldots, n+m$, which proves that $\vec{\delta_o}$ satisfies the Kuhn–Tucker conditions. ◇

1) Optimal Delay Assignment Algorithm: In order to find a solution to the optimization problem, we first note that a new flow request is rejected if $\sum_{i=1}^{K} l_i > \Delta$. The algorithm *Opt()*, outlined in Fig. 1, ensures that whenever the new flow request can be accepted by the network, it is accepted with the smallest possible increase in the network load.

The input parameters of the algorithm are the delay bounds l_i and u_i, for each node $i = 1, \ldots, K$, along the routing path, and the maximum end-to-end delay requirement Δ of the new flow. Using these values, the algorithm distributes Δ optimally such that the delay, δ_i, at each node i starting with the source node minimizes $\sum_{i=1}^{K} (1/\delta_i)$, while satisfying the constraints $\sum_{i=1}^{K} \delta_i = \Delta$ and $l_i \leq \delta_i \leq u_i$ for $i = 1, \ldots, K$.

The following three lemmas, which can be proven by simple algebraic manipulation, capture the variations and bounds on the value of ξ in the *Opt()* algorithm.

$$\sum_{j \in I} \lambda_j \nabla g_j(\vec{\delta_o}) = [-\lambda_1 \quad \ldots \quad -\lambda_n \quad \lambda_{K+n+1} \quad \ldots \quad \lambda_{K+n+m} \quad 0 \quad \ldots \quad 0]^T. \quad (11)$$

Lemma 1: If $u_i \le \xi_k$ and $\xi_{k+1} = (\xi_k n_f - u_i)/(n_f - 1)$, then $\xi_{k+1} \ge \xi_k$. That is, when a node is added to UF in step 2, the value of ξ increases. ◇

Lemma 2: If $l_j \ge \xi_k$ and $\xi_{k+1} = (\xi_k n_f - l_j)/(n_f - 1)$, then $\xi_{k+1} \le \xi_k$. That is, when a node is added to LF in step 3, the value of ξ decreases. ◇

Lemma 3: If $u_i > \xi_k$ and $\xi_{k+1} = (\xi_k n_f + u_i)/(n_f + 1)$, then $\xi_k < \xi_{k+1} < u_i$. That is, when a node is removed from UF in step 3, the value of ξ increases but stays smaller than u_i. ◇

Next, we prove in Lemma 4 that step 2 of algorithm *Opt()* adds to UF every node i with u_i smaller than or equal to the equi-partitioned value of the nodes that are not in UF. We then prove in Lemma 5 that, after each iteration of step 3 in *Opt()*, UF contains exactly every node i with u_i smaller than or equal to the equi-partitioned value, and that every node, j in LF has a value l_j larger or equal to the equi-partitioned value. We finally use Lemma 6 to prove that at the end of step 3, LF contains exactly every node j with a value l_j larger than or equal to the equi-partitioned value. The complexity of the *Opt()* algorithm is proven in Theorem 2.

Lemma 4: If the "while" loop in step 2 of *Opt()* terminates with $k = k_1$, then a node q is in UF_{k_1} if and only if $u_q \le \xi_{k_1}$. In other words, $\max_{q \in UF_{k_1}}\{u_q\} \le \xi_{k_1}$ and any q with $u_q \le \xi_{k_1}$ is in UF_{k_1}.

Proof: First, note that $LF = \emptyset$ in step 2. Consider a node i which is added to UF_k when $k = a$. Hence $u_i \le \xi_a$. From Lemma 1, the addition of any node to UF_k at any step k leads to $\xi_{k+1} \ge \xi_k$. That is the addition of a node to UF increases the value of ξ. Hence, $\xi_a \le \xi_{k_1}$ which leads to $u_i \le \xi_{k_1}$. This is true for any node i that is added to UF and thus $\max_{q \in UF_{k_1}}\{u_q\} \le \xi_{k_1}$. Moreover, the while loop in step 2 does not stop until every node with $u_q \le \xi_k$ is added to UF. Hence, any q with $u_q \le \xi_{k_1}$ is in UF_{k_1}. ◇

After constructing UF in step 2, the algorithm *Opt()* constructs LF in step 3. In each iteration of step 3, a node, j, with l_j larger or equal to the equi-partitioned value is added to LF. After each node is added to LF, the equi-partitioned value becomes smaller (see Lemma 2) and thus some nodes that are in UF may have their upper bound larger than the equi-partitioned value. The inner loop in step 3 (the Repeat loop) removes such nodes from UF.

Lemma 5: Assume that a given iteration of the while loop in step 3 is entered with $k = k_2$ and terminates with $k = k_3$, where $k_3 > k_2$, then the following holds:

- a node i is in UF_{k_3} if and only if $u_i \le \xi_{k_3}$,
- for any node $j \in LF_{k_3}$, $l_j \ge \xi_{k_3}$.

Proof: Assume that the conditions of the Lemma are satisfied at $k = k_2$. That is the following hypotheses are satisfied:

$$\mathbf{H1}: \min_{j \in LF_{k_2}} \{l_j\} \ge \xi_{k_2}$$
$$\mathbf{H2}: \max_{i \in UF_{k_2}} \{u_i\} \le \xi_{k_2} \text{ and}$$
$$\mathbf{H3}: \text{any } i \text{ with } u_i \le \xi_{k_2} \text{is in } UF_{k_2}.$$

In an iteration of step 3, a node j with $l_j \ge \xi_{k_2}$ is added to LF_{k_2}. From Lemma 2, this leads to $\xi_{k_2+1} \le \xi_{k_2}$. Moreover, in the Repeat loop in step 3, when a node i is removed from UF_a for some $k_2 < a < k_3$, we have by Lemma 3 $\xi_a \le \xi_{a+1} \le u_i$. But $u_i \le \xi_{k_3}$ from $\mathbf{H2}$. Hence, we obtain

$$\xi_{k_2+1} \le \xi_{k_3} \le \xi_{k_2}.$$

Based on the above, and using $\mathbf{H1}$ and $l_j \ge \xi_{k_2}$, we obtain the following:

$$\mathbf{C1}: \min_{j \in LF_{k_3}} \{l_j\} \ge \xi_{k_3}.$$

Given that the Repeat loop in each iteration of step 3 does not stop unless every node i in UF_k has $u_i \le \xi_k$, we have

$$\mathbf{C2}: \max_{i \in UF_{k_3}} \{u_i\} \le \xi_{k_3}.$$

Finally, assume that for some node i, $u_i \le \xi_{k_3}$ and yet $i \notin UF_{k_3}$. Given that $\xi_{k_3} \le \xi_{k_2}$ then $i \in \xi_{k_2}$. Hence, i should have been removed from UF in the Repeat loop. Assume that i is removed from UF_a for some $k_2 < a < k_3$. Because $u_i = \max_{q \in UF_a}\{u_q\}$, then $u_i \ge u_q$ for any other node q which is removed from UF_b for $a \le b \le k_3$. From Lemma 3, the removal of q leads to $\xi_{b+1} < u_q \le u_i$. In other words, $\xi_{k_3} < u_i$ which is a contradiction with the assumption that $u_i \le \xi_{k_3}$. Hence, we obtain the following:

$$\mathbf{C3}: \text{any } i \text{ with } u_i \le \xi_{k_3} \text{ is in } UF_{k_3}.$$

We have shown that if the conditions of the Lemma are satisfied when an iteration of step 3 starts ($\mathbf{H1}$, $\mathbf{H2}$, and $\mathbf{H3}$) then the conditions are satisfied when the iteration ends ($\mathbf{C1}$, $\mathbf{C2}$, and $\mathbf{C3}$). Clearly, for the first iteration of step 3, $k_2 = k_1$ where k_1 is the value of k at the end of step 2. Lemma 4 and the fact that $LF_{k_1} = \emptyset$ at the end of step 2 proves that $\mathbf{H1}$, $\mathbf{H2}$, and $\mathbf{H3}$ are satisfied at the start of the first iteration of step 3. Moreover, if $\mathbf{C1}$, $\mathbf{C2}$, and $\mathbf{C3}$ are satisfied at the end of an iteration of step 3, then $\mathbf{H1}$, $\mathbf{H2}$, and $\mathbf{H3}$ are satisfied at the beginning of the next iteration. Hence, by induction, $\mathbf{C1}$, $\mathbf{C2}$, and $\mathbf{C3}$ are satisfied at the end of each iteration of step 3. ◇

Lemma 6: At the end of the second loop (the end of the algorithm), a node i is in UF if and only if $u_i \le \xi_k$ and a node j is in LF if and only if $l_j \ge \xi_k$.

Proof: The proof is straight forward from Lemma 5 and the fact that step 3 of the algorithm does not stop until all nodes with $u_j \ge \xi$ are included in LF. ◇

Theorem 2: Algorithm Opt() finds a solution which satisfies the conditions of Theorem 1 in $O(K)$ time~complexity

Proof: By construction, the algorithm always updates ξ such that $\xi = \left(\Delta - \sum_{i \in LF} l_i - \sum_{i \in UF} u_i\right)/n_f$ and always maintains $LF \cap UF = \emptyset$. Hence by including in NF the nodes that are not in $UF \cup LF$, and from Lemma 6, we conclude that the algorithm finds a solution which satisfies the condition of Theorem 1.

The first loop (step 2) executes at most K times since at most K nodes can be added to UF. The second loop (step 3) also executes at most K times since at most K nodes can be added to LF. Moreover, each iteration of the inner loop in step 3 (the Repeat loop) removes at most one node from UF, while no nodes are added to UF after the first loop (step 2). Hence, during step 3, at most K nodes are removed from UF. The total algorithm complexity is thus $O(K)$. ◇

C. Case of Heterogeneous Networks

In the case of a heterogeneous network, the routing nodes operate at different rates, $\mu_i \neq \mu_j,: 1 \leq i,j \leq K$. Consequently, the per-node delay assignment problem can be expressed as: **Minimize** $\sum_{i=1}^{K}(\mu_i/\delta_i)$ Subject to $\sum_{i=1}^{K}\delta_i \leq \Delta$, $l_i \leq \delta_i$, and $\delta_i \leq u_i$.

Note that if $\sum_{i=1}^{K} l_i > \Delta$ no per-node delay assignment is feasible and the connection request should be rejected. It can also be noted that if $\sum_{i=1}^{K} l_i = \Delta$ then the optimal solution is $\delta_i = l_i$ for all nodes $i(1 \leq i \leq K)$, across the routing path, since increasing δ_i for any node i causes an end-to-end delay violation. In the following, we assume that $\sum_{i=1}^{K} l_i < \Delta$, and we proceed to derive a solution to the per-node delay assignment problem in a heterogeneous network.

It should be noted that the requested maximum end-to-end delay, Δ, should be used in its entirety in order to optimize the objective function. In other words, assume that there exists a set of values $\delta_i : (1 \leq i \leq K)$, which verify the constraints $l_i \leq \delta_i \leq u_i$ and $\sum_{i=1}^{K}\delta_i < \Delta$. Then it is easy to show that there exist a set of $\hat{\delta}_i$ and a set of ϵ_i, such that $\hat{\delta}_i = \delta_i + \epsilon_i$, $\sum_{i=1}^{K}\hat{\delta}_i = \Delta$; and $\hat{\delta}_i(1 \leq i \leq K)$ further minimize the objective function, thereby negating the optimality of δ_is. This is due to the decreasing nature of $(\mu_i/\delta_i)(1 \leq i \leq K)$. Based on this observation and using the fact that $\delta_i \geq l_i$, for all $i = 1, \ldots K$, the per-node delay optimization problem can be expressed as: **Minimize** $\sum_{i=1}^{K}(\mu_i/\hat{\delta}_i + l_i)$, **subject to:** $\sum_{i=1}^{K}\hat{\delta}_i = \hat{\Delta}, 0 \leq \hat{\delta}_i$ and $\hat{\delta}_i \leq \hat{u}_i = u_i - l_i$, where $\hat{\Delta} = \Delta - \sum_{i=1}^{K} l_i$, $\hat{\delta}_i = \delta_i - l_i$, and $\hat{u_i} = u_i - l_i$.

To solve the above problem, referred to as Opt_LU, we adopt a technique similar to the one described in [1]. More specifically, we first introduce and solve two auxiliary optimization problems: the first, referred to as Opt_E, considers only the equality constraint, while the second, referred to as Opt_L, takes into consideration the equality and lower bound constraints, but ignores the upper bound constraints. The solutions of these two problems will be used to solve the original per-node delay assignment problem formalized above.

1) Opt_E Solution: The Opt_E problem does not take into account the boundary constraints, and thus can be expressed as: **Minimize** $\sum_{i=1}^{K}(\mu_i/\hat{\Delta}_i + l_i)$, **subject to** $\sum_{i=1}^{K}\hat{\Delta}_i = \hat{\Delta}$.

The application of Lagrange multipliers technique to the above problem yields

$$-\frac{\mu_i}{(\hat{\Delta}_i + l_i)^2} + \lambda = 0 \quad i = 1, 2, \ldots, K \tag{15}$$

where λ is the Lagrange multiplier. Using the fact that $\sum_{i=1}^{K}\hat{\delta}_i = \hat{\Delta}$, $\hat{\delta}_i \geq 0$, and (15), results in

$$\delta_i = \frac{\sqrt{\mu_i}}{\sum_{j=1}^{K}\mu_j} \cdot \Delta. \tag{16}$$

2) Opt_L Solution: The Opt_L problem can be expressed as

$$\textbf{maximize}: \quad -\sum_{i=1}^{K}\frac{\mu_i}{\hat{\delta}_i + l_i} \tag{17}$$

$$\textbf{Subject to}: \quad \sum_{i=1}^{K}\hat{\delta}_i = \hat{\Delta} \tag{18}$$

$$0 \leq \hat{\delta}_i \quad i = 1, 2, \ldots, K. \tag{19}$$

To solve Opt_L, we first evaluate the solution set S_{Opt_L} to the corresponding problem Opt_E and check whether all inequality constraints are automatically satisfied. If this is the case, the solution set S_{Opt_L} of the Opt_L problem reduces to the solution set, S_{O_Lpt}. Otherwise, S_{Opt_L} will be constructed iteratively as described below.

A well-known result of nonlinear optimization theory states that the solution S_{Opt_L} of Opt_L must satisfies *Kuhn–Tucker conditions* [17]. Furthermore, Kuhn–Tucker conditions are also sufficient due to the properties of the objective function. For Problem Opt_L, Kuhn–Tucker conditions can be derived from (18) and (19) as

$$-\frac{\mu_i}{(\hat{\delta}_i + l_i)^2} + \lambda - \beta_i = 0 \quad i = 1, 2 \ldots K \tag{20}$$

$$-\beta_i\hat{\delta}_i = 0 \quad i = 1, 2 \ldots K \tag{21}$$

$$\beta_i \geq 0 \quad i = 1, 2 \ldots K \tag{22}$$

where $\lambda, \beta_1, \ldots, \beta_K$ are Lagrange multipliers. The necessary and sufficient character of Kuhn–Tucker conditions indicates that any $2K + 1$ tuple $(\hat{\delta}_1, \ldots, \hat{\delta}_K, \beta_1, \ldots, \beta_K, \lambda)$ which satisfies conditions (18)–(22) provides optimal $\hat{\delta}_i$ values for Opt_L.

One method for solving the optimization problem Opt_L is to find a solution to the $2K + 1$(18), (20), and (21) which satisfies constraint sets (19) and (22). Iteratively solving the $2K + 1$ nonlinear equations is a complex process which is not guaranteed to converge. A more efficient approach to the solution uses the Kuhn–Tucker conditions (20)–(22) to prove some useful properties of the optimal solution. The properties derived are then used to *refine* the solution of the optimization problem Opt_E. These properties are captured in the following lemmas.

Lemma 7: If S_{Opt_E} violates some inequality constraints given by (19), then $\exists i$ such that $\beta_i > 0$.

Proof: Assume to the contrary that $\forall\, i\beta_i = 0$. In this case, Kuhn–Tucker conditions reduce to the equality constraint (18), the set of inequality constraints (19) plus the Lagrangian condition given in (14). On the other hand, the set S_{Opt_E} should satisfy (18) and the Lagrangian condition (15). In other words, solving Opt_E is always equivalent to solving a set of nonlinear equations which are identical to Kuhn–Tucker conditions of Opt_L, *except fo the inequality constraints*, by setting $\beta_i = 0$, for all $(1 \leq i \leq K)$. Hence, if there were a solution to Opt_L where for all i $\beta_i = 0$, then the solution will be discovered by Opt_E algorithm described above without the occurrence of any inequality constraint violations. This is in contradiction with the assumption that the solution S_{Opt_E} fails to satisfy all the inequality constraints. Therefore, there exists at least one Lagrange multiplier β_i that is strictly greater than 0. ◇

Lemma 8: If S_{Opt_E} violates some inequality constraints given by (19), then $\exists j$ such that $\beta_j = 0$.

Proof: Assume that $\forall\, j, \beta_j > 0$. In this case, (21) implies that $\forall\ i\ \hat{\delta}_i = 0$. This implies that $\sum_{i=1}^{K} \hat{\delta}_i$ is equal to 0, and the delay budget, $\hat{\Delta}$, remains totally unused. This violates the optimality property of the solution. ◇

Let $\mathcal{F} = \{f_i() \mid f_i(\hat{\delta}_i) = (\mu_i/\hat{\delta}_i + l_i)\}$ and consider the set of indices $\mathcal{I}$, defined as follows:

$$\mathcal{I} = \{m \mid -f'_m(0) \leq -f'_i(0)\, \forall\, i\}. \tag{23}$$

Considering $f_i()$s as reward functions, $f'_m(\hat{\delta}_i)$ can be viewed as the marginal return associated with $f_m(\hat{\delta}_i)$. The set $\mathcal{I}$ then contains the indices of the functions $f_m()$, such as $-f_m()$ leads to the smallest marginal returns at the lower bound 0.

Lemma 9: If S_{Opt_E} violates some inequality constraints then, in S_{Opt_L}, $\hat{\delta}_m = 0, \forall\, m \in \mathcal{I}$.

Proof: Assume that $\exists m \in \mathcal{I}$ such that $\hat{\delta}_m > 0$. In this case, (21) implies that the corresponding $\beta_m = 0$. Based on Lemma (7), $\exists j$ such that $\beta_j > 0$. Based on (21), $\hat{\delta}_j = 0$. Using (20), leads to $-f'_m(\hat{\delta}_m) + f'_j(0) = \beta_j > 0$. Since $\hat{\delta}_m > 0$, the property of $f_m()$ suggests that $-f'_m(\hat{\delta}_m) \geq -f'_m(0)$. But in this case, we obtain $-f'_m(0) + f'_j(0) \leq \beta_j > 0$. This contradicts the assumption that $m \in \mathcal{I}$. Hence $\beta_m > 0$ and by (22) $\hat{\delta}_m = 0$. ◇

The result of Lemma (9) can be used to develop the following iterative algorithm:

```
Algorithm Opt_L($\mathcal{F}, \hat{\Delta}$)
1. Discard the inequality
 constraints and solve for $S_{Opt\_E}$.
Check the feasibility of $S_{Opt\_E}$
2. If all the inequality constraints
 are satisfied then $S_{Opt\_L} = S_{Opt\_E}$; exit
3. Compute $\mathcal{I}$ as described in (23)
4. Set $\hat{\delta}_m = 0\ \forall\ m \in \mathcal{I}$
5. Set $\mathcal{F} = \mathcal{F} - \mathcal{I}$
6. Go to step 1
```

3) Opt_LU Optimal Solution: Opt_LU is characterized by the set $\mathcal{F} = \{f_i() | f_i(\hat{\delta}_i)\} = (\mu_i/\hat{\delta}_i + l_i)$, the set $\hat{\mathcal{U}} = \{\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_K\}$ of upper bounds, and the end-to-end delay budget $\hat{\Delta}$. The optimization problem can be expressed as

$$\textbf{maximize}: \quad -\sum_{i=1}^{K} f_i(\hat{\delta}_i) \tag{24}$$

$$\textbf{Subject to}: \quad \sum_{i=1}^{K} \hat{\delta}_i = \hat{\Delta} \tag{25}$$

$$\hat{\delta}_i \leq \hat{u}_i \quad i = 1, 2, \ldots K \tag{26}$$

$$0 \leq \hat{\delta}_i \quad i = 1, 2, \ldots K. \tag{27}$$

Furthermore, we have $0 < \hat{\Delta} < \sum_{i=1}^{K} \hat{u}_i$ and $0 < \hat{u}_i\ \forall\ i$. Opt_L differs from Opt_LU in the additional set of upper bound constraints. Consequently, it is easy to show that if S_{Opt_L} satisfies the constraints given by (26), the set S_{Opt_L} is a feasible solution for Opt_LU, and $S_{Opt\_L} = S_{Opt\_LU}$. However, if an upper bound constraint is violated, an iterative process, in a way analogous to the process used to derive S_{Opt_L}, must be used to remove upper bound constraint violations.

```
Algorithm Opt_LU($\mathcal{F}, \hat{\mathcal{U}}, \hat{\Delta}$)
 1.  Set $S_{Opt\_LU} = \emptyset$
 2.  If $\mathcal{F} = \emptyset$ then exit
 3.  Find $S_{Opt\_L}$ by invoking Algorithm Opt_L
 4.  If all upper bounds are satisfied then
       $S_{Opt\_LU} = (S_{Opt\_LU} \cup S_{Opt\_L})$; exit
 5.  Else Compute $\mathcal{G}$
 6.  Set $\hat{\delta}_q = \hat{u}_q \forall q \in \mathcal{G}$ in $S_{Opt\_LU}$
 7.  Set $\hat{\Delta} = \hat{\Delta} - \sum_{m \in \mathcal{G}} \hat{u}_m$
 8.  Set $\mathcal{F} = \mathcal{F} - \mathcal{G}$
 9.  Set $\hat{\mathcal{U}} = \hat{\mathcal{U}} - \{\hat{u}_k | k \in \mathcal{G}\}$
10.  Go to step 2
```

Fig. 2. Opt_LU algorithm.

Let $\mathcal{G} = \{m | -f'_m(\hat{u}_m) \geq -f'_i(\hat{u}_i),\ \forall\, i\}$. The set $\mathcal{G}$ contains the functions $f_m \in \mathcal{F}$, such that $-f'_m()$ leads to the largest marginal returns at the upper bounds.

The algorithm *Opt_LU()*, depicted in Fig. 2, solves the Opt_LU problem based on successive invocations of *Opt_L()*. First, we find the solution S_{Opt_L} of the corresponding Opt_L problem. The solution, if it exists, is optimal for the Opt_L problem, which does not take into account upper bound constraints. If the upper bound constraints are automatically satisfied, S_{Opt_L} is also optimal for the Opt_LU problem. However, if this is not the case, we first set $\hat{\delta}_q = \hat{u}_q\ \forall\ q \in \mathcal{G}$. We then update the sets $\mathcal{F}, \hat{\mathcal{U}}$ and the end-to-end delay budget, $\hat{\Delta}$, before we proceed with the next iteration.

The correctness of the algorithm can be argued in a similar fashion as in the case of the Opt_L problem. Deriving the necessary and sufficient Kuhn–Tucker conditions for problem Opt_LU, results in

$$-f'_i(\hat{\delta}_i) + \lambda + \bar{\beta}_i - \beta_i = 0 \quad i = 1, 2, \ldots, K \tag{28}$$

$$\bar{\beta}_i(\hat{\delta}_i - \hat{u}_i) = 0 \quad i = 1, 2, \ldots, K \tag{29}$$

$$-\beta_i\hat{\delta}_i = 0 \quad i = 1, 2, \ldots, K \tag{30}$$

$$\bar{\beta}_i \geq 0 \quad i = 1, 2, \ldots, K \tag{31}$$

$$\beta_i \geq 0 \quad i = 1, 2, \ldots, K \tag{32}$$

where $\lambda, \bar{\beta}_1, \ldots, \bar{\beta}_n, \beta_1, \ldots, \beta_K$ are Lagrange multipliers.

It can easily be shown that if S_{Opt_L} violates upper bound constraints given by (26) then $\exists i \bar{\beta}_i > 0$. A similar argument, which states that if $\forall\ i \bar{\beta}_i > 0$ then $\beta_i = 0$, can also be proven. These two observations can then be used to prove that if the Lagrange multipliers $\bar{\mu}_i$, $i \in \mathcal{G}$, are all nonzero, this implies, based on (29), that $\hat{\delta}_q = \delta_q, \forall\ q \in \mathcal{G}$.

D. Delay Assignments Heuristics

One possible approach to compute suitable per-node delay values considers the processing capabilities of the node as the limiting factor, assuming that each node has a relatively large amount of buffers. These initial values are then adjusted to meet the buffering requirements of each node along the routing path without violating the end-to-end delay requirements of the flow. This policy is likely to achieve a more efficient use of the network resources, which in turn increases the capability of a node to support future flow requests. The overhead such a policy entails, however, may be prohibitive when the network load is light and the benefit of reducing the network resource consumption is

```
Heuristic EPH()
1. for $i$=1 to $K$
    • $\Delta_i^1 = l_i \cdot \frac{\Delta}{\sum_{i=1}^{K} l_i}$
    • $\Delta_i^2 = l_i +$ abs($u_i -$ ($l_i + l_{i-1}$))
    • $\Delta_i$=min($\Delta_i^1, \Delta_i^2$)
2. for $i$=1 to $K$
    • while $\Delta_i > u_i$
        * $\Delta_i = l_i$ + ($\Delta_i$ - $l_i$)/2.0
3. while $\sum_{i=1}^{K} \Delta_i > \Delta$
    • for $i$=1 to $K$
        * $\Delta_i = l_i + (\Delta_i - l_i)/2.0$
```

Fig. 3. Equi-partition heuristic.

```
Heuristic LBH()
1. for $i$=1 to $K$
    • $\Delta_i^1$ = max( $\Delta \cdot \frac{\rho_i}{\sum_{i=1}^{K} \rho_i}, l_i$ )
    • $\Delta_i^2 = l_i +$ abs($u_i -$ ($l_i + l_{i-1}$)
    • $\Delta_i$=min( $\Delta_i^1, \Delta_i^2$ )
2. for $i$=1 to $K$
    • while $\Delta_i > u_i$
        * $\Delta_i = l_i$ + ($\Delta_i$ - $l_i$)/2.0
3. while $\sum_{i=1}^{K} \Delta_i > \Delta$
    • for $i$=1 to $K$
        * $\Delta_i = l_i$ + ($\Delta_i$ - $l_i$)/2.0
```

Fig. 4. Load balancing heuristic.

small. A different approach would be to balance the load across the routing path in order to minimize the likelihood bottlenecks. To achieve this goal, the policy assigns larger per-node delay budgets to highly loaded nodes than to lightly loaded nodes. This is based on the observation that assigning a large delay value to a given flow causes a relatively small load increase to heavily loaded node. In the following, we describe two heuristics, namely equi-partition heuristic, *EPH()*, and load balancing heuristic, *LBH()*. *EPH()* aims at reducing the processing load placed by the new flow over the node, while *LBH()* attempts to distribute the load uniformly across the routing path. The performance of these two heuristics are then compared to the optimal policy.

1) Equi-Partition Heuristic: The basic steps of the heuristic are described in Fig. 3. The input parameters of *EPH()* include the delay bounds, l_i and u_i for each node $i = 1, \ldots, K$ along the routing path, derived from the new flow traffic rate specification and the end-to-end delay requirement Δ of the underlying application.

The approach used by *EPH()* is to compute two potential delay values, Δ_i^1 and Δ_i^2, for each node i along the routing path. The first value, Δ_i^1, is computed based on the assumption that each node has a relatively large amount of excess buffer capacity, and only the processing capacity needs to be considered. This is achieved by taking the routing path's excess delay and distributing it proportionally across all nodes on the routing path. Therefore, Δ_i^1 is set to be equal to $l_i \cdot (\Delta / \sum_1^K l_i)$. Increasing the delay value a routing node would support reduces the effective processing capacity requirements of the node at the expense of increasing its buffer requirements. Notice that the sum over i of Δ_i^1s does not exceed the new flow's end-to-end delay requirement, Δ. It is not always the case, however, that Δ_i^1 is feasible at each node along the routing path. Due to the limited number of buffers currently available, a node may not be capable to support its assigned value Δ_i^1. To address this limitation, *EPH()* computes a second potential delay value Δ_i^2, based on the assumption that the number of buffers available at node i is small, thereby limiting the range of supportable delay values. Since node i must provide enough buffers to potentially hold the new flow's packets for at least l_{i-1} units of time in their waiting room, a potential delay bound for the new flow at node i can be set equal to $u_i + l_{i-1}$. Therefore, a potential delay value that can satisfy both the processing and buffer capacities of a node can be computed as $\Delta_i^2 = l_i + (u_i - (l_i + l_{i-1}))$.

A delay value, Δ_i, which appropriately addresses the node's excess processing and buffer capacities, can be set equal to the minimum of Δ_i^1 and Δ_i^2. If Δ_i, $i = 1, \ldots, K$ is not feasible along the routing path, the procedure attempts to adjust these values by considering that l_i and Δ_i as the end-points of a search interval and attempting to locate a value in this interval that satisfies the feasibility requirement of each node along the routing path. The search continues, as described in steps 2 and 3 in Fig. 3, until a feasible value is determined.

2) Load Balancing Heuristic: *LBH()* attempts to balance the load along the routing path when accepting a flow request. It does so by computing the initial lower bound delay values to be proportional to the nodes' respective loads and then adjusting these values such that they lie within each node's smallest and largest feasible delay values, without violating the end-to-end delay requirement of the flow. Based on this strategy, a lightly loaded node is assigned a smaller delay value and thus takes on a higher load, while a highly loaded node is assigned a higher delay value and sees a smaller increase in its load.

In addition to the smallest and largest delay values, l_i and u_i, supported by each node $i = 1 \ldots K$ along the routing path and the maximum end-to-end delay requirement, Δ, of the new flow, *LBH()* includes the current load, ρ_i, at node i as input parameter. The heuristic uses the information about the current workloads of the nodes to achieve a balanced delay assignment across the routing path. The basic steps of the heuristic are described in Fig. 4.

Initially, *LBH()* computes a lower bound delay value Δ_i^1 to be proportional to the load of each node along the routing path. If this value is not feasible, *LBH()* attempts to adjust this value to meet the processing and buffer capacities of each node along the routing path. The procedure used by *LBH()* to adjust the initial lower bound delay value assigned to each node is similar to the one used in *EPH()*. An upper bound delay value Δ_i^2 is computed based exclusively on the node's buffer constraints. If the minimum of the two values Δ_i^1 and Δ_i^2 is not feasible, a search procedure to locate a feasible value within the interval $[l_i, \Delta_i]$ is initiated. The search continues until a feasible value is determined.

V. Simulation Results and Analysis

In order to assess the performance of the proposed per-node delay assignment schemes, several simulation-based studies were conducted. This is achieved by simulating the behavior of a general topology composed of switching nodes that are configured with different resource capabilities. The performance metrics of the simulation include network resource utilization and the flow acceptance ratio.

In the studies discussed in this paper, the simulated network was comprised of 20 nodes with identical buffering and processing capacities. The network topology was selected so that: 1) it ensures pure randomness in the way the characteristics of the incoming flows, along with their arrival and their departure times, were selected; 2) it allows a large number of simulation runs to be conducted in order to eliminate any probable biased behavior that could occur during the analysis; and 3) it reduces the complexity of computing the shortest paths between any two nodes so the impact of routing on the results is minimized. The final topology selected can be logically viewed as a bidirectional ring connecting all nodes in the network.

In all experiments, the profiles of the simulated flows, in terms of traffic specification, were generated randomly. The end-to-end delay requirements for each flow was also generated randomly. However, to accommodate a wide spectrum of flow requirement specifications, three types of end-to-end delay ranges were considered. Each range of delays reflects how tight is the end-to-end delay requirement relative to the range of delays supported by a node along the routing path. Three cases where considered, namely *stringent*, *moderate*, and *relaxed*. In the stringent case, the end-to-end delay requirements of the flows were randomly generated from within an interval of $[0, 100]$ msec. In the moderate case, the delay interval was extended to $[0, 350]$, while in the *relaxed* case, the delay interval was increased to $[0, 500]$.

For each new flow request, a source and a destination node were selected randomly and the shortest path connecting these two nodes was computed. This path and the profile of the new flow request were then used by *Opt()* and the heuristics *EPH()* and *LBH()*, in three different instances of the simulation, to determine the feasibility of establishing or rejecting the new flow request. The simulation results have been obtained for two models, namely *Static* and *Dynamic*. In the Static model, the optimal algorithm and the two heuristics are compared in a scenario where the flows last for the lifetime of the simulation experiment. In the Dynamic model, however, flows are characterized by their average inter-arrival rate and their average service duration. In the following, the results of the optimal algorithm and the two heuristics are presented and discussed. In order to nullify biasing, the value of each point in the graphs discussed below represents the average value of a 100 simulation runs.

A. Static Model Results

Fig. 5 shows the average number of flows accepted against the number of flows generated for the *Opt()* algorithm and the two heuristics. In all three cases, the results show that when the network load is low, the number of accepted flows increases rapidly with the number of flows generated. As the network reaches saturation, however, the number stabilizes. As expected, the results show that the *Opt()* algorithm outperforms the two heuristics. The performance improvement of the *Opt()* algorithm over *EPH()*, however, is not as significant as its improvement over *LBH()*.

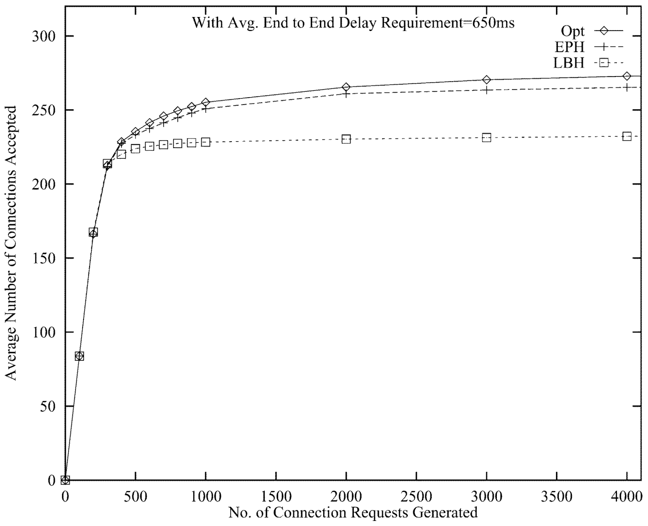

Fig. 5. Average number of successful connections.

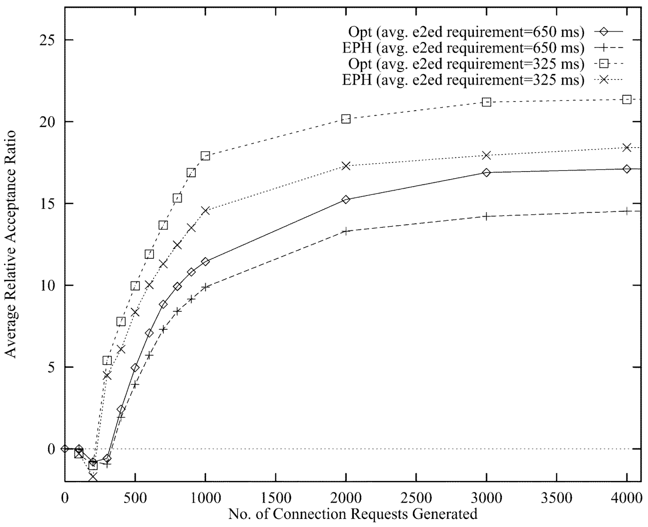

Fig. 6. *Opt()* and *EPH()* acceptance ratio relative to *LBH()*.

In order to compare the effectiveness of minimizing the additional load imposed on the network by the new connection (the goal of *Opt()* and *EPH()*) and the effectiveness of balancing the load of the new connection among the routing nodes (the goal of *LBH()*), we define the relative acceptance ratio of *Opt()* as the difference between the number of connections accepted by *Opt()* and the number of connections accepted by *LBH()* relative to the number of connections accepted by *LBH()*. The relative acceptance ratio of *EPH()* is defined similarly.

Figs. 6 and 7 depict the average relative acceptance ratio of *Opt()* and *EPH()* against the number of flows generated. The results indicate that when the network load is low, *LBH()* performs 1% to 4% better than *Opt()* and 2% to 5% better than *EPH()*. This is due to the fact that the processing and buffering capacities gains *Opt()* and *EPH()* achieve is not significant when

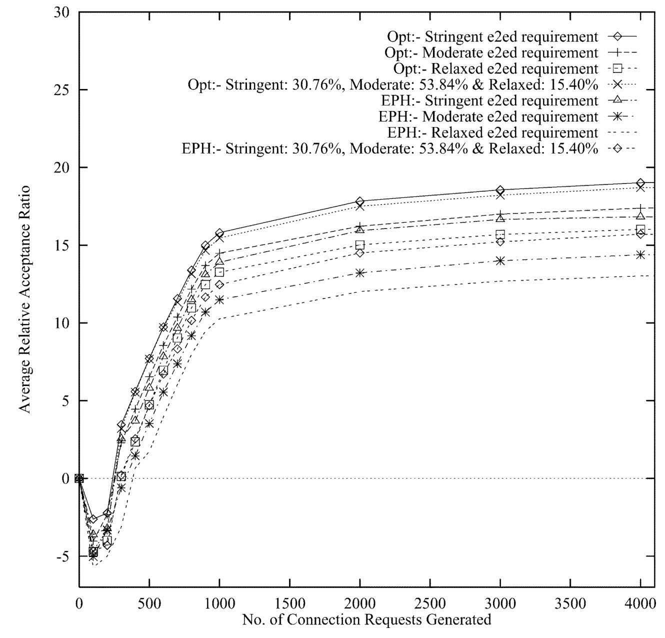

Fig. 7. Effect of end-to-end delay requirements.

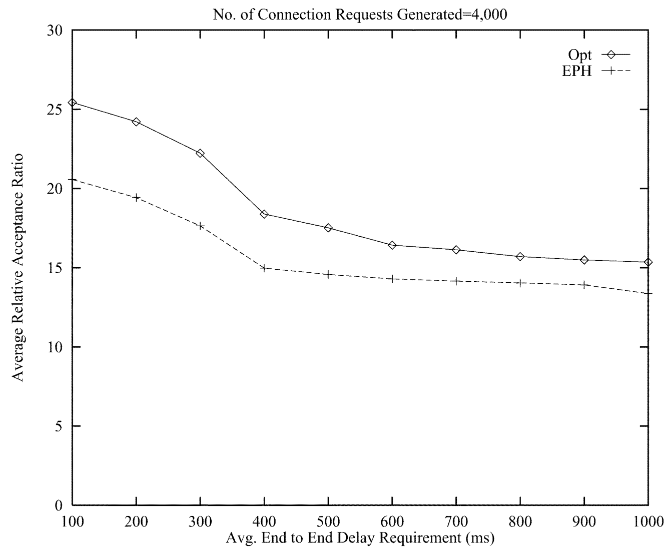

Fig. 8. Effect of end-to-end delay near network saturation.

the network load is low. *LBH()*, on the other hand, manages to achieve a more balanced load among the routing nodes, which results in a higher flow acceptance ratio in comparison to *Opt()* and *EPH()*. However, as the network load becomes high, the gains of *Opt()* and *EPH()* become significant and result in a higher flow acceptance ratio in comparison to *LBH()*. These gains range from 15% to 20% for *EPH()*, and from 15% to 20% for *LBH*.

What also emerges indirectly from these graphs and more evidently from Fig. 8 is that at high network load, the average relative acceptance ratio of *Opt()* and *EPH()* increases with tighter end-to-end delay requirements of the flows. This is due to the fact that when the end-to-end delay requirements of the accepted flows are tight, the load imposed on the network is more stringent than when end-to-end delay requirements are moderate. Consequently, when the QoS requirements are stringent, the

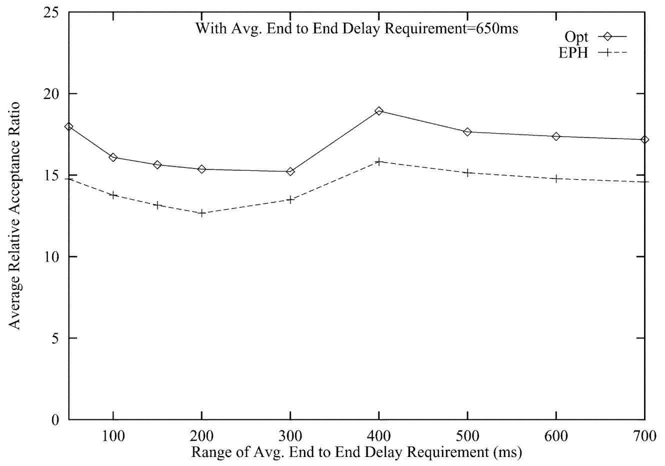

Fig. 9. Effect of the range of the end-to-end delay.

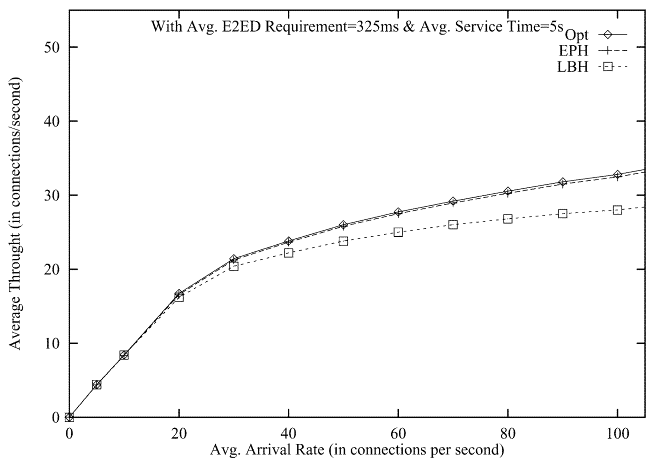

Fig. 10. Average throughput of the dynamic model.

per-flow intrinsic gains achieved by *Opt()* and *EPH()*, in terms of reducing the network load, is higher than the gain achieved by it *LBH()*. This enables *Opt()* and *EPH()* to accept a greater percentage of flows with smaller end-to-end delay requirements than *LBH*.

Fig. 9 shows the average relative acceptance ratio against the range of average end-to-end delay requirement at high network load (4000 requests). The results indicate that at small ranges, the load balancing feature is incorporated more in *Opt()* and *EPH()* than at moderate ranges thereby enabling them in accepting a greater percentage of flows at smaller ranges. However, as the range of the end-to-end delay requirement increases, the load balancing feature of *LBH()* diminishes because of the larger variation in the end-to-end delay requirements of the incoming flows. Thus, at higher ranges, the acceptance rate of *Opt()* and *LBH* increases more than at moderate ranges and remains more or less the same at even higher ranges.

B. Dynamic Model Results

In this case, flows are generated dynamically. Each flow remains active for a certain period of time, after which the flow terminates and releases all the resources across the routing path. The results of the this study are depicted in Figs. 10–12, respectively.

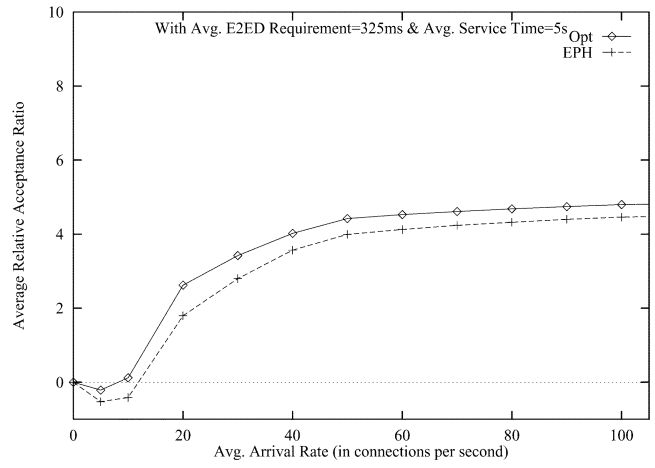

Fig. 11. Relative acceptance ratio of the dynamic model.

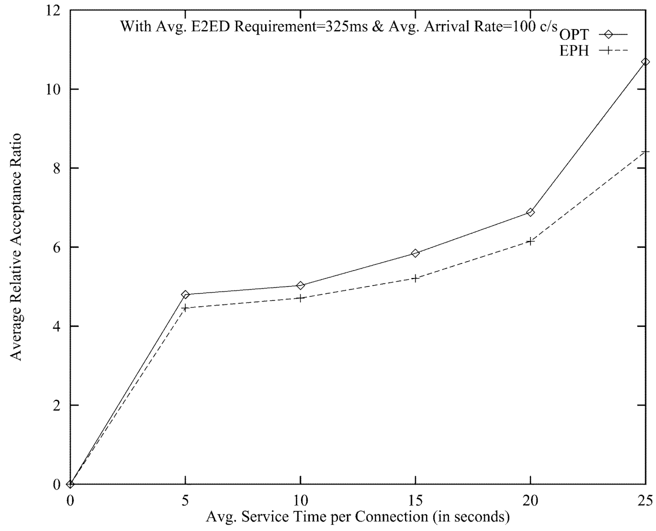

Fig. 12. Effect of average service time.

Fig. 10 shows the average throughput for *Opt()*, *EPH()* and *LBH()* against the average arrival rate of the flows. As the figure indicates, the increase in the average throughput with the average arrival rate is more pronounced in *LBH()* than in *Opt()* and *EPH()* at low arrival rate (low network load). *Opt()* and *EPH()* perform better than *LBH()* at high network loads.

Fig. 11 shows the average relative acceptance ratio of *Opt()* and *EPH()* (over *LBH()*) against the average arrival rate of the flows. Similar to the static model, *LBH()* performs better than *Opt()* (by less than 0.5) and *EPH()* (by less than 1) when the network load is low. As the arrival rate increases, similar to the case of a static model, *Opt()* and *EPH()* perform better than *LBH()* when the network load becomes high. The performance improvement of *Opt()* is around 5, while *EPH()* outperforms it LBH() by 4.5.

Fig. 12 depicts the average relative acceptance ratio of *Opt()* and *EPH()* over *LBH()* against the average flow service time, for an average connection arrival rate of 100 connections per second. As is expected, the acceptance ratio increases with the increase in the average service time. This happens because greater service time of the flows implies that these flows continue to stay in the network occupying the resources of the network for longer durations of time. Therefore, before they depart, more new flows come and are either accepted or rejected by the network. If accepted, then, they will also be occupying the network resources for longer periods of time. Consequently, the network load builds up and as explained before, this favors *Opt()* and *EPH()* to perform increasingly better than *LBH()* with increasing network load.

VI. Conclusion

In this paper, we proposed a methodology to compute feasible delay values for different classes of scheduling strategies. We also described an optimal algorithm and two heuristics which can be used to assign feasible delay values so that a specific objective is achieved. A set of simulation experiments were developed and used to compare the performance of each scheme.

The results show that the optimal algorithm produces higher flow acceptance ratios than the two other schemes. For lightly loaded networks, however, the results show that the computational complexity of the optimal algorithm may not be warranted and simple heuristics usually lead to highly acceptable results.

Acknowledgment

The authors would like to thank A. Vagish for his help in conducting the simulation experiments. The authors also wish to gratefully acknowledge H. Aydin for his useful comments.

References

[1] H. Aydin, “Enhancing performance and fault tolerance in reward-based scheduling,” Ph.D. dissertation, Dept. of Comp. Sci., Univ. Pittsburgh, Pittsburgh, PA, 2001.
[2] J. C. R. Bennett and H. Zhang, “Why WFQ is not good enough for integrated services networks?,” in *Proc. NOSSDAV’96*, April 1996.
[3] D. C. Clark, S. Shenker, and L. Zhang, “Supporting real-time applications in an integrated services packet network: Architecture and mechanism,” in *Proc. SIGCOMM’92*, 1992, pp. 14–26.
[4] R. L. Cruz, “Service burstiness and dynamic burstiness measures: A framework,” *J. High Speed Networks*, vol. 2, pp. 105–127, 1992.
[5] C. Dovrolis, D. Stiliadis, and P. Ramanathan, “Proportional differentiated services: Delay differentiation and packet scheduling,” *IEEE Trans. Commun.*, vol. 10, pp. 12–26, Feb. 2002.
[6] B. Field, T. F. Znati, and D. Mosse, “V-net: A framework for a versatile network architecture to support real-time communication performance guarantees,” in *Proc. IEEE INFOCOM*, Apr. 1995, pp. 1188–1196.
[7] V. Firoiu, J. Kurose, and D. Towsley, “Efficient admission control for edf schedulers,” in *Proc. IEEE INFOCOM*, Apr. 1997, pp. 310–317.
[8] P. Goyal and H. M. Vin, “Fair airport scheduling algorithms,” in *Proc. NOSSDAV*, St. Louis, MO, May 1997.
[9] A. Hung and G. Kesidis, “Bandwidth scheduling for wide-area ATM networks using virtual finishing times,” *IEEE/ACM Trans. Networking*, vol. 4, pp. 49–54, Feb. 1996.
[10] C. R. Kalmanek, H. Kanakia, and S. Keshav, “Rate controlled servers for very high-speed networks,” in *Proc. IEEE GlobeCom*, 1990, pp. 12–20.
[11] E. W. Knightly, R. F. Mines, and H. Zhang, “Deterministic characterization and network utilizations for several distributed real-time applications,” in *Proc. IEEE WORDS*, Dana Point, CA, Oct. 1994, pp. 63–70.
[12] R. Guerin, L. Georgiadis, and A. Parekh, “Optimal multiplexing on a single link,” *IEEE Trans. Inform. Theory*, vol. 43, pp. 1518–1535, Sept. 1997.
[13] Y.-C. Lai and W.-H. Li, “A novel scheduler for proportional delay differentiation by considering packet transmission time,” *IEEE Commun. Lett.*, vol. 7, pp. 189–191, Apr. 2003.
[14] J. Liebeherr, D. Wrege, and D. Ferrari, “Exact admission control for networks with bounded delayyservices,” *IEEE/ACM Trans. Networking*, vol. 4, pp. 885–901, Dec. 1996.

[15] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. Assoc. Comput. Machin.*, vol. 20, pp. 46–61, Jan. 1973.

[16] D. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.

[17] M. J. Panik, *Classical Optimization: Foundations and Extensions*. Amsterdam, The Netherlands: Elsevier, 1976.

[18] A. Parekh, "A generalized processor sharing approach ti flow control in integrated services networks," Ph.D. dissertation, Laboratory for Information and Decision Systems, Massachusetts Insititute of Technology, 1992.

[19] H. Sariowan, R. L. Cruz, and G. C. Polyzos, "SCED: A generalized scheduling policy for guaranteeing quality-of-service," *IEEE/ACM Trans. Networking*, vol. 7, pp. 669–684, Oct. 1999.

[20] I. Stoica, H. Zhang, and T. S. E. Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time and priority services," presented at the Proc. ACM SIGCOMM, Cannes, France, 1997.

[21] S. Suri, G. Varghese, and G. Chandranmenon, "Leap forward virtual clock: A new fair queueing scheme with guaranteed delays and throughput fairness," Washington Univ., St. Louis, MO, Tech. Rep. WUCS-96-10, 1996.

[22] D. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," in *Proc. TriComm*, 1991, pp. 35–43.

[23] D. E. Wrege, E. W. Knightly, H. Zhang, and J. Liebeherr, "Deterministic delay bounds for VBR video in packet-switching networks: Fundamental limits and practical trade-offs," *IEEE/ACM Trans. Networking*, vol. 4, pp. 352–362, June 1996.

[24] G. G. Xie and S. S. Lam, "An efficient adaptive search algorithm for scheduling real-time traffic," presented at the IEEE ICNP, Columbus, OH, Oct. 1996.

[25] L. Zhang, "A new architecture for packet switching network protocols," Ph.D. dissertation, Dept. of Elect. Eng. and Comp. Sci., Massachusetts Institute Technology, 1989.

Taieb F. Znati (A'91) received the M.S. degree in computer science from Purdue University, Lafayette, IN, in 1984, and the Ph.D. degree in computer science from Michigan State University, Lansing, in 1988.

He is currently a Professor in the Department of Computer Science, with a joint appointment in Telecommunications in the Department of Information Science, University of Pittsburgh, Pittsburgh, PA. His current research interests focus on the design of network protocols for wired and wireless communication networks to support applications' QoS requirements.

Dr. Znati currently serves as General Chair of IEEE INFOCOM 2005, General Chair of SECON 2004, the first IEEE conference on Sensor and Ad Hoc Communications and Networks, General Chair of the Annual Simulation Symposium, and General Chair of the Communication Networks and Distributed Systems Modeling and Simulation Conference. He is a member of the Editorial Board of the *International Journal of Parallel and Distributed Systems and Networks*, a member of the Editorial Board of the *Journal on Wireless Communications and Mobile Computing*, the *Journal on Ad-Hoc Networks*, and a member of IEEE Transactions of Parallel and Distributed Systems, and *Wireless Networks*, the journal of mobile communication, computation and information. He is currently serving as a Senior Program Director for networking research at the National Science Foundation.

Rami Melhem (F'00) received the B.E. degree in electrical engineering from Cairo University, Cairo, Egypt, in 1976, the M.A. degree in mathematics and the M.S. degree in computer science from the University of Pittsburgh in 1981, and the Ph.D. degree in computer science from the University of Pittsburgh, Pittsburgh, PA, in 1983.

He was an Assistant Professor at Purdue University, Lafayette, IN, prior to joining the faculty of the University of Pittsburgh in 1986, where he is currently a Professor of Computer Science and Electrical Engineering and the Chair of the Computer Science Department. His research interest include real-time and fault-tolerant systems, optical networks, high-performance computing, and parallel computer architectures.

Dr. Melhem served on program committees of numerous conferences and workshops and was the General Chair for the Third International Conference on Massively Parallel Processing Using Optical Interconnections. He was on the editorial board of the IEEE TRANSACTIONS ON COMPUTERS and the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is serving on the advisory boards of the IEEE technical committees on parallel processing and on computer architecture. He is the editor for the Kluwer/Plenum Book Series in Computer Science and is on the editorial board of the *Computer Architecture Letters*, the *International Journal of Embedded Systems*, and the *Journal of Parallel and Distributed Computing*. He is a member of the ACM.