On Centralized Smooth Scheduling

Dedicated to the memory of Professor Shimon Even for his inspiration and encouragement

Ami Litman* Shiri Moran-Schein*

Abstract

This paper studies evenly distributed sets of natural numbers and their applications to scheduling in a centralized environment. Such sets, called *smooth sets*, have the property that their quantity within each interval is proportional to the size of the interval, up to a bounded additive deviation; namely, for $\rho, \Delta \in \mathbb{R}$ a set A of natural numbers is (ρ, Δ)-*smooth* if $\text{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any interval $I \subset \mathbb{N}$.

The current paper studies scheduling *persistent clients* on a single slot-oriented *resource* in a flexible and predictable manner. Each client γ has a given *rate* ρ_γ that defines the share of the resource he is entitled to receive and the goal is a *smooth schedule* in which, for some predefined Δ, each client γ is served in a (ρ_γ, Δ)-smooth set of slots (natural numbers).

The paper considers a centralized environment where a single algorithm computes the user of the current slot. (An accompanying paper studies a distributed environment in which each client by itself computes whether or not it owns the current slot.) An important contribution of this paper is the construction of a smooth schedule with an extremely efficient algorithm that computes the user of each slot in $O(\log \log q)$ time and $O(n)$ space, where n is the number of clients and $q \triangleq \max\{\rho_\gamma/\rho_{\gamma'} \mid \gamma, \gamma' \in \Gamma\}$; in many practical applications this $O(\log \log q)$ value is actually a small constant.

Our scheduling technique is based on a reduction from allocation of slots to allocation of sub-intervals of the unit interval. This technique naturally extends to the problem of scheduling multiple resources, even under the restriction that a client can be served concurrently by at most one resource. This paper constructs such a schedule in which the users of each slot are computed extremely fast — in $O(m \log \log q)$ time per slot and $O(n)$ space where m is the number of resources; this result is a significant improvement on the prior fastest algorithm that produces such a schedule (actually of a special type — a P-fair schedule) in $O(m \log n)$ time per slot and $O(n)$ space.

Moreover, the paper introduces a novel approach to multi-resource scheduling in which each resource independently computes, slot after slot, what client to serve in this slot; the paper presents such a schedule computed in $O(\log \log q)$ time per slot and $O(n)$ space; prior to our work, nothing was known about such independent computation. Finally, this paper demonstrates the usefulness of smooth schedules by showing that they are highly attractive for multiplexing the links of a connection-oriented packet switching network.

*Department of Computer Science, Technion, Haifa 32000, Israel. E-mail: {litman,mshiri}@cs.technion.ac.il.

1 Introduction

1.1 Smooth Scheduling

This paper studies evenly distributed sets of natural numbers and their applications to scheduling in a centralized environment. Such sets, called *smooth sets*, have the property that their quantity within each interval is proportional to the size of the interval, up to a bounded additive deviation; namely, for $\rho, \Delta \in \mathbb{R}$ a set A of natural numbers is (ρ, Δ)-*smooth* if $\mathrm{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any interval $I \subset \mathbb{N}$; a set A is Δ-smooth if it is (ρ, Δ)-smooth for some real number ρ. An earlier paper of us [22] establish the concept of smooth sets and the current paper builds on the mathematical infrastructure constructed there; it also builds on tools and techniques developed in an accompanying paper [21] which studies applications of smooth sets to scheduling in a distributed environment.

As demonstrated in this paper and in [21], smooth sets are highly attractive for scheduling *persistent clients* [7] having pre-defined rates on a single slot-oriented *resource* in a flexible and predictable manner. In this framework, time is divided into discrete *slots* and in each slot the resource (e.g., a communication channel) may serve at most one *client* (e.g., a session). Each client γ has a pre-defined *rate* $\rho_\gamma \in [0, 1]$ that defines the share of the resource he is entitled to receive. A *smooth schedule* for such a problem is a schedule in which, for some predefined Δ, each client γ is served in a (ρ_γ, Δ)-smooth set of slots (natural numbers). Such a schedule enjoys the following two attractive properties. **Proper rate** — the average amount of service received by a client γ in the long-run is a ρ_γ fraction of the resource. **Bounded deviation** — the number of slots a client γ receives during any k consecutive slots deviates from his nominal share of $k \cdot \rho_\gamma$ by less than a pre-specified constant.

We apply a novel approach to scheduling in which the scheduling process is divided into two stages. In the *allocation stage* each client is allocated an infinite set of slots that is generated via abstract mathematical operations. In the *online stage* an algorithm computes, slot after slot, the client which is served in this slot. This contrast most previous approaches in which the allocation is just a by-product of the online algorithm. Moreover, the mathematical operations we use enable exceedingly fast online algorithms working in, essentially, a constant time per slot. (The allocation stage is also efficient and takes polynomial time.) This contrast most other algorithms that produce smooth schedules (e.g., [8, 9, 26]) as they are based on priority queues and their time per slot is at least logarithmic in the size of the queues.

The current paper studies a centralized environment in which a single algorithm computes, slot after slot, the owner of this slot. In contrast, our accompanying paper [21] studies a distributed environment in which each client by itself (without any inter-client communication) computes, slot after slot, whether or not it owns this slot.

An important contribution of the current paper is the construction, under some reasonable restrictions, of a smooth schedule in which the owner of each slot is computed extremely fast, essentially in a constant time. Our scheduling technique is based on a reduction from allocation of slots to allocation of sub-intervals of the unit interval. This technique naturally extends to the problem of scheduling multiple resources, even under the restriction that a client can be served concurrently by at most one resource. Our extended technique constructs such a schedule in which the users of each slot are computed extremely fast in time that is independent of the number of clients. This result is a significant improvement on the prior fastest algorithm [9] that produces such a schedule (actually of a special type — a P-fair schedule). Moreover, the paper introduces a novel approach to multi-resource scheduling in which each resource independently (without any inter-resource communication) computes the user of each slot. Another important contribution of this paper is the construction of such a schedule in which each resource computes the user of each of its slots in essentially a constant time per slot. Finally, the paper demonstrates the usefulness of smooth schedules by showing that they are highly attractive for multiplexing the links of a connection-oriented packet switching network.

1.2 Dispatching in Constant Time

We henceforth identify the slots of the resource with the natural numbers and use the following notations. For a set A of slots (natural numbers), if the following limit, $\lim_{|I|\to\infty}(|A \cap I|/|I|)$, exists where I ranges over the intervals of natural numbers, then this limit is called the *rate* of A and is denoted *rate*(A). Clearly, a (ρ, Δ)-smooth set has rate ρ. Let Γ be a finite set of *clients*. A Γ-

system $\{X_\gamma \mid \gamma \in \Gamma\}$ is an association of an element X_γ with each $\gamma \in \Gamma$. A Γ-*allocation* is a Γ-system $\{A_\gamma \subset \mathbb{N} \mid \gamma \in \Gamma\}$ of disjoint sets of natural numbers. Such an allocation is Δ-smooth if every A_γ is Δ-smooth; the allocation is *smooth* if it is Δ-smooth for some Δ. A *rate function* ρ *over* Γ is a function that assigns to each $\gamma \in \Gamma$ a real number $\rho_\gamma \in [0,1]$. For such a ρ, a ρ-*allocation* is a Γ-allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ with $\mathrm{rate}(A_\gamma) = \rho_\gamma$.

A *dispatcher* of a Γ-allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ is an algorithm that *dispatches* the slots (natural numbers) one by one. That is, for each integer i in its turn the dispatcher determines the owner of this integer — the client $\gamma \in \Gamma$ s.t. $i \in A_\gamma$ — or determines that there is no such γ. A dispatcher operates on a RAM of a predefined width; such a dispatcher is of *space* s and *time* t if it uses at most s memory words and dispatches each slot in at most t time units.

The first result concerning smooth allocation is due to Tijdeman [26] who proved that any rate function, in which the sum of the rates is at most one, has a 2-smooth allocation. Tijdeman did not address the dispatching complexity; however, since his algorithm is based on a priority argument, the time complexity of the dispatching algorithm seems to be no better than $O(\log n)$, where n is the number of clients.

An important contribution of this paper is that any rate function, obeying some reasonable restrictions, has a smooth allocation with two extremely efficient dispatchers whose time is either constant or essentially constant, as stated by the following two theorems. In these theorems a *k-bit rational* is a number which is the ratio of two k-bit numbers. Our model of computation is the popular RAM [13], and the *k-bit RAM* is the RAM with k-bit words. (Such a RAM performs standard operations on these words in constant time.)

Theorem 3: Let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \leq 0.99$ and all the ρ_γ are k-bit rational. Let $q = \max\left\{\rho_\gamma/\rho_{\gamma'} \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth ρ-allocation which has two dispatchers on any $\Omega(k)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space[1] and the second of $O(1)$ time and $O(n \cdot q)$ space.

(The numbering of this theorem, as well as of the other theorems stated in the introduction, follows that of the main part of the paper.) The schedule of Theorem 3 utilizes only 99% of the resource. This utilization can be improved to 100% if we impose a certain restriction on the rates. To this end, a *k-bit fraction* is a (non-negative) number having a binary expansion with k bits to the right of the binary point; a number is of l *significant-bits* if its binary expansion has an interval of l bits s.t. all the bits outside this interval are 0. In other words, the former number is of the form $j/2^k$ for some $j \in \mathbb{N}$ and the latter number is of the form $i \cdot 2^j$ with $j \in \mathbb{Z}$, $i \in \mathbb{N}$ and $i < 2^l$.

Theorem 2: Let ρ be a rate function over Γ s.t. $\sum_\gamma \rho_\gamma \leq 1$ and all the ρ_γ are k-bit fractions of l significant-bits. Let $q = \max\left\{\rho_\gamma/\rho_{\gamma'} \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is an $(l+1)$-smooth ρ-allocation which has two dispatchers on any $\Omega(k)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space and the second of $O(1)$ time and $O(n \cdot q)$ space.

Prior to this work, the fastest dispatchers of a smooth schedule [26, 9] worked in $O(\log n)$ time and $O(n)$ space on the same RAM of the above theorems. More about the second work [9] follows shortly.

1.3 Multiple Resources

A natural extension of our work concerns scheduling of m identical resources, where the sum of the rates of the clients is (of course) at most m and the rate of each client is at most one. If it is permitted to serve a single client by several resources in one time-slot then it is easy to transform any scheduling technique of a single resource into a scheduling technique of m identical resources, as indicated in [8]; see also Section 4. However, in some applications (e.g., CPU scheduling) a client can be served concurrently by at most one resource. As pointed out by Liu [23] and again by Baruah at el. [8] "the simple fact that a client can use only one resource even when several resources are free at the same time adds a surprising amount of difficulty" (to the scheduling problem).

[1] The bounds on the number of memory words and their width are critical; without one of them it is a simple matter to encode the entire schedule in a finite number of words and to retrieve the owners of the slots, one by one, in constant time.

Surprisingly, it turns out that under our scheduling technique this is not the case. Namely, a straightforward extension of our technique produces the following theorems, which provides extremely efficient dispatchers for this problem whose time is either $O(m)$ or essentially $O(m)$. Prior to our work, the fastest dispatcher for the problem [9] was of $O(m \log n)$ time where n is the number of clients.

We use the following terminology in the context of m identical resources[2]. For a set of clients Γ, a (Γ, m)-*allocation* is a Γ-system $\{A_\gamma \subseteq \mathbb{N} \mid \gamma \in \Gamma\}$ s.t. for any slot $i \in \mathbb{N}$, the set of clients served in this slot, $\{\gamma \mid i \in A_\gamma\}$, has at most m clients. Note that the allocation specifies which clients are served in each slot but does not specify which client each resource serves. For a rate function ρ over Γ, a (ρ, m)-allocation is a (Γ, m)-allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ with $\mathrm{rate}(A_\gamma) = \rho_\gamma$. A *dispatcher* of such an allocation is an algorithm that computes, for each slot i, the set of clients that are served in this slot (but does not specify which client each resource serves).

Theorem 5: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma \in \Gamma} \rho_\gamma \leq 0.99 \cdot m$, each $\rho_\gamma \leq 1$ and all the ρ_γ are k-bit rationals. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth (ρ, m)-allocation which has two dispatchers on any $\Omega(k + \log n)$-bit RAM: the first of $O(m \log\log q)$ time and $O(n)$ space, and the second of $O(m)$ time and $O(n \cdot q)$ space.

Again, the utilization can be improved to 100% under a certain restriction on the rates, as follows.

Theorem 4: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma \in \Gamma} \rho_\gamma \leq m$, each $\rho_\gamma \leq 1$ and all the ρ_γ are k-bit fraction of l significant-bits. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is an $(l+1)$-smooth (ρ, m)-allocation which has two dispatchers on any $\Omega(k+\log n)$-bit RAM: the first of $O(m \log\log q)$ time and $O(n)$ space and the second of $O(m)$ time and $O(n \cdot q)$ space.

Theorems 4 and 5 consider a $\Omega(k + \log n)$-bit RAM. The same expression is also implicit in Theorems 2 and 3 since the premise of these theorems implies that $k = \Omega(\log n)$.

Independent Dispatching: In some applications (e.g., CPU scheduling), each resource has its own computational capabilities. In this case it is highly desirable that each resource independently dispatches its own slots; that is, in each slot the resource computes what client to serve. As stated formally by the following theorem, our scheduling technique achieves this *independent dispatching* with dispatchers whose time per-slot does not depend on the number of clients or on the number of resources, and is either constant or essentially constant. Moreover, the service provided by any single resource to each of the clients is 10-smooth. Prior to our work, nothing was known about a smooth schedule with efficient independent dispatchers and nothing was known about such schedule in which the service provided by each resource is smooth.

The following theorem, which summarizes our independent dispatching, uses the following terminology. A sequence $A^1, \cdots, A^m$ of Γ-allocations *realizes* a (Γ, m)-allocation A if when each resource j serves the clients as per the allocation A^j then the combined service is as per the (Γ, m)-allocation A; namely, no slot is assigned to the same client by two distinct A^j allocations and in each time-slot i the set $\left\{\gamma \mid \exists j : \; i \in A^j_\gamma\right\}$ of clients served by the combined allocation in this time-slot equals to the set of clients served by the (Γ, m)-allocation A in that time-slot.

Theorem 7: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma \in \Gamma} \rho_\gamma \leq 0.99 \cdot m$, each $\rho_\gamma \leq 1$ and all the ρ_γ are k-bit rationals. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth (ρ, m)-allocation which is realized by a sequence $A^1, A^2, \cdots A^m$ of Γ-allocations s.t. each of the A^j is 10-smooth and has two dispatchers on any $\Omega(k + \log n)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space, and the second of $O(1)$ time and $O(n \cdot q)$ space.

The problem of scheduling multiple resources under the restriction of no concurrent service was resolved in 1993 by Baruah et al. in a seminal paper [8]. Actually, they introduced and solved an harder problem, the *P-fair periodic scheduling problem*, in which each client γ has to be scheduled during the first t slots either $\lceil \rho_\gamma \cdot t \rceil$ or $\lfloor \rho_\gamma \cdot t \rfloor$ times. In [9] Baruah at el. presented the currently fastest dispatcher of a P-fair schedule and this dispatcher is of $O(m \log n)$ time and $O(n)$ space on the same RAM as ours. Many works study variants and extensions of the P-fair scheduling problem, e.g., [9, 1, 24, 18, 12, 4, 2, 3, 5, 11].

[2] This terminology is different from that of [21] due to the different focus of the current paper.

A P-fair schedule is a special form of 2-smooth schedule [22]. Thus, the P-fair problem differs from our problem, solved in Theorem 5, in two main aspects: the sum of the rates of the clients can be as large as m (rather than $0.99 \cdot m$) and the schedule is 2-smooth (rather than 10-smooth). In many applications, however, the difference between a 2-smooth schedule and a 10-smooth schedule is not essential; an example of such an application, concerning connection-oriented packet switching network, is given in Section 5.

The main advantage of our technique over the known P-fair scheduling techniques is its exceedingly fast dispatchers. Another advantage of our technique, summarized in Theorem 7, is that each resource can independently dispatch its own slots in a time that is essentially constant. Nothing is known about efficient independent dispatchers of a P-fair schedule. Moreover, nothing is known about any efficient parallel implementation of a dispatcher of a P-fair schedule.

The above theorems are phrased in an existential manner. However, they summarize our scheduling technique that actually constructs the allocations and the dispatchers in question and, moreover, all these objects can be constructed in polynomial time. In addition, the two dispatchers provided by each of our theorems actually represent the endpoints of a wide spectrum of dispatchers with time-space tradeoff. Finally, in Theorems 3, 5 and 7 the number 0.99 can be replaced with $1 - \epsilon$ where $\epsilon > 0$. In this case the resulting allocation is $O(\log(1/\epsilon))$-smooth while the other parameters in the conclusion of the theorems are intact.

1.4 Scheduling Communication Networks

The issue of scheduling the links of a connection-oriented packet-switching network has been studied extensively; see the enlightening survey of Zhang [28]. The current paper propose an attractive new scheme for scheduling the links of such a network, based on Theorem 3. This scheme is an example of an application of smooth scheduling in which the difference between a 2-smooth schedule and a 10-smooth schedule is not essential.

We assume the standard model of packet switching network [19]. That is, the network is made of nodes (switches) connected by directed links; time is divided into discrete time-slots (which are identified with the natural numbers); in each slot a link transmits either one packet or no packet. In the simple model, the transmission delay of a link is one time-slot; however, we consider a more realistic model in which each link has a fixed delay which is an integral number of slots. (Nevertheless, in each time-slot the link can start transmitting a new packet.) Switches do not introduce any delay; that is, a packet that entered a switch during a certain slot is ready to be transmitted during the following one.

In our case the network is used by *connections* (a.k.a. *sessions*) where each connection is a sequence of packets (a *packets stream*) that traverses a predefined path from its *source* to its *destination*; the packets of a packet-stream follow each other (and never get lost); in each time-slot the connection initiates at most one packet; and each connection γ has a *rate* ρ_γ which constrains its bandwidth as follows: In any k consecutive slots the connection initiates at most $\rho_\gamma \cdot k + \Delta'$ packets, where Δ' is some global constant. Many works assume such a constraint [28, 14, 15, 25, 16, 10, 6, 17].

In our scheme, each link is considered in isolation, without regard to the other links. Its slots are pre-allocated to the connections using it so that, for some (small) constant Δ, the slots allocated to each connection constitute a Δ-smooth set with the given rate. Any schedule that is based on a pre-allocation is, of course, non-greedy (a.k.a. non work-conserving). That is, a link may be idle while packets are ready to cross it. Moreover, our scheme has a second level of flow control — a link may be idle in a slot while packets of the connection having this slot are ready to cross it.

The performance of a networking scheme is usually measured by the following parameters [28]. Two parameters, which are associated with each connection, are the (worst case) end-to-end delay of the connection and its buffering requirements. Under our scheme, the end-to-end delay of a connection γ is $O(l_\gamma/\rho_\gamma) + \sum_{i=1}^{l_\gamma} d_i$, when l_γ is the length of the path of γ and d_i is the delay of the i-th link of the path. The buffering requirements of a connection γ is $O(1)$ per link; thus, the buffering requirements per link are $O(n)$, when n is the number of the connections using the link. Another parameter associated with a connection is its jitter [28]. Our scheme guarantees zero jitter; this, of course, requires additional buffers which are already included in the above $O(n)$ buffering requirements.

An efficient networking scheme should serve concurrently a large set of connections, and therefore the restrictions it imposes on those connections should be as weak as possible. Our scheme can schedule any set of connections obeying the only restriction that the sum of the rates of the connections crossing any given link is at most 0.99. We say that the *capacity* of such a scheme is 0.99 and clearly the maximal achievable capacity is 1.

It is highly desirable [28] that the online algorithms controlling the links are very fast. Our algorithms are extremely fast — their time per slot of a link does not depend on the number of connections using the link and is essentially constant.

Our scheme compares favorably with prior networking schemes. There is no scheme that outperforms ours in **all** the above parameters and most of the schemes that outperform ours in some of these parameters do so only by a small constant factor.

A notable exception is the outstanding result of Andrews at el. [6]; they proved the existence of a schedule which is, in many aspects, optimal. The capacity of their scheme is one — it schedules any set of connections obeying the only restriction that the sum of the rates of the connections crossing any given link is at most one. In their schedule the end-to-end delay of a connection γ is just $O(1/\rho_\gamma + l_\gamma)$, when l_γ is the length of the path associated with the connection. (This is under the assumption that the transmission delay of a link is one time-slot.) Their scheme distinguishes between two classes of buffers. Buffers of the first class are used to store packets that just entered the system and have not crossed any link, yet; their scheme requires $O(1)$ such buffers per connection. Buffers of the second class are used to store packets in transit; their scheme requires only $O(1)$ such buffers per link; this is in contrast to most of the networking schemes [28], including ours, which require $O(1)$ buffers per link per connection. Thus, w.r.t. the delay and the buffering requirements their work significantly outperforms ours, as well as most of the other works in this field [28]. The weakness of their result relates to the complexity of the online computation. Their paper [6] does not address this complexity which seems to be exceedingly high, and thus it seems that their scheme is impractical. Another weakness of their scheme is that it does not guarantee zero jitter. As said, in our scheme a link dispatches each slot in essentially a constant time while using moderate amount of memory and our scheme guarantees zero jitter.

The rest of this paper is organized as follows. Section 2 presents notions and lemmas from [22, 21] which are used here. Section 3 constructs smooth schedules with extremely efficient dispatchers, Section 4 extends our technique for scheduling multiple resources, and Section 5 presents an attractive scheme for scheduling the links of connection oriented packet switching network.

2 Imported Tools

As said, this paper builds on the mathematical infrastructure of smooth sets developed in [22] and on fast resolution of smooth sets developed in [21]. This section reviews notions and lemmas from these papers which are used here.

Smooth Sets Recall that a set $A \subset \mathbb{N}$ is (ρ, Δ)-*smooth* if $\text{abs}(|I| \cdot \rho - |I \cap A|) < \Delta$ for any interval I of $\mathbb{N}$.

Lemma 1: [22] Let $A_1, A_2 \subset \mathbb{N}$ be (ρ_1, Δ_1)-smooth and (ρ_2, Δ_2)-smooth, respectively. Then:

a. If $A_1 \cap A_2 = \emptyset$ then $A_1 \cup A_2$ is $(\rho_1 + \rho_2, \Delta_1 + \Delta_2)$-smooth.

b. The set $\mathbb{N} \setminus A_1$ is $(1 - \rho_1, \Delta_1)$-smooth.

c. If $A_2 \subset A_1$ then $A_1 \setminus A_2$ is $(\rho_1 - \rho_2, \Delta_1 + \Delta_2)$-smooth.

Lemma 2: [20, 22] A $(\rho, 1)$-smooth set $A \subset \mathbb{N}$ exists for any $0 \le \rho \le 1$.

To address the time and space complexity of our algorithms, we assume the popular *RAM* model of computation [13]. Specifically, the memory of a *k-bit RAM* is organized in binary *words* of k bits, each having a distinct k-bit address and such a RAM performs standard operations on these words in constant time.

Resolution of Sets A *resolver* of a set of slots (natural numbers) is an algorithm that determines, slot after slot, whether the slot belongs or does not belong to the set. A resolver operates on a RAM of a predefined width; such a resolver is of *space* s and *time* t if it uses at most s memory words and resolves each slot in at most t time units.

Lemma 3: [21] Any 1-smooth set whose rate is a k-bit rational has a resolver of $O(1)$ time and $O(1)$ space on any $\Omega(k)$-bit RAM.

Composition of Sets For $A, B \subset \mathbb{N}$ the *set composition* of A and B is the subset D of A s.t., for all i, the i-th member of A is in D iff the i-th member of $\mathbb{N}$ is in B and A has an i-th member. In other words, the composition of A and B is $A \circ B \triangleq \{i \in A \ : \ |[0, i) \cap A| \in B\}$. Note that for any $D \subset A$ there is a set $B \subset \mathbb{N}$ s.t. $D = A \circ B$ and, when A is infinite, this B is unique.

Lemma 4: [22] Composition of sets is an associative operator.

Lemma 5: [22] Let A_1 and A_2 be (ρ_1, Δ_1)-smooth and (ρ_2, Δ_2)-smooth subsets of $\mathbb{N}$. Then:

a. $A_1 \circ A_2$ is $(\rho_1 \cdot \rho_2, \rho_2 \cdot \Delta_1 + \Delta_2)$-smooth.

b. If $\Delta_2 = 1$ and $\rho_2 \leq (\Delta_1 - 1)/\Delta_1$ then $A_1 \circ A_2$ is $(\rho_1 \cdot \rho_2, \Delta_1)$-smooth.

A Shuffle The scheduling technique of the paper is based on a one-to-one partial function from the unit interval into the natural numbers s.t. an interval of length ρ is mapped onto a subset of the natural numbers with rate ρ. To formalize this, we henceforth extend any function f to be defined on any set Z (not necessarily a subset of the domain of f) by $f(Z) \triangleq \{f(z) \mid z \in Z \ \wedge \ f(z) \text{ is defined}\}$. For a real interval Z, let $\|Z\|$ denote the length of Z. A *shuffle* is a partial one-to-one function f from the unit interval into the natural numbers s.t. for any interval $X \subset [0, 1)$, $\text{rate}(\mu(X)) = \|X\|$.

Theorem 1: [22] There is a shuffle f s.t. for any $0 < \epsilon$ and for any real interval $X \subset [0, 1)$, there exists an interval $Y \subset X$ s.t. $\|X\|(1 - \epsilon) \leq \|Y\|$ and $f(Y)$ is $O(\log(1/\epsilon))$-smooth.

Note that the length of the above X intervals, as well as their endpoints, are not necessarily rational.

The shuffle, satisfying Theorem 1, constructed in [22] is denoted μ and called the *infinite bit reversal function* (and is related to the finite bit reversal permutation). This μ is defined only on *binary-fractions* — numbers of the form $l/2^j$ with $j, l \in \mathbb{N}$ — in the unit interval. Let $x \in [0, 1)$ be a binary-fraction and let $0.\alpha_0\alpha_1\alpha_2\cdots$ be its binary expansion. Then $\mu(x) \in \mathbb{N}$ is the number whose binary expansion is $\cdots\alpha_2\alpha_1\alpha_0$. This definition is meaningful since $\alpha_i \neq 0$ for finitely many i.

Our technique employs Lemma 6 below which uses the following terminology. Let $n, j \in \mathbb{N}$. An $(n \times j)$*-interval* is an interval $X \subset [0, 1)$ of the form $X = [l/2^j, (l + n)/2^j)$ for some $l \in \mathbb{N}$. We allow one or two of the parameters, n and j, to be replaced with a $*$ which stands for any natural number. For example, a $(* \times j)$*-interval* is an interval that is an $(n \times j)$-interval for some n (including the case of $n = 0$). An *extreme interval* is either the empty interval or a $(* \times *)$-interval X s.t. for some $(1 \times *)$-interval Y, $X \subset Y$ and X shares an endpoint with Y.

Lemma 6: [22] Let X be a non-empty $(n \times *)$-interval. Then:

a. The set $\mu(X)$ is $(\lceil \log n \rceil + 1)$-smooth.

b. Either X is an extreme interval or there are $\Delta_1, \Delta_2 \in \mathbb{R}$ and two disjoint extreme intervals X_1 and X_2 s.t. $X_1 \cup X_2 = X$, $\mu(X_1)$ is Δ_1-smooth, $\mu(X_2)$ is Δ_2-smooth and $\Delta_1 + \Delta_2 = (\lceil \log n \rceil + 1)$.

For $x \in \mathbb{R}$ and $j \in \mathbb{N}$, let $\lfloor x \rfloor^j$ denote the maximal j-bit fraction which is not larger than x. In other words, $\lfloor x \rfloor^j$ is the number obtained from the binary expansion of x by deleting all the bits rightwards to, and including, the $(j+1)$-th position after the point. For $j, n \in \mathbb{N}$, let $\nu^j(n) \triangleq \lfloor \mu^{-1}(n) \rfloor^j$.

Lemma 7: [21] For any $j \in \mathbb{N}$, the infinite sequence $\langle \nu^j(0), \nu^j(1), \ldots \rangle$ can be generated in $O(1)$ space and $O(1)$ time per element on any $\Omega(j)$-bit RAM.

3 Dispatching in Constant Time

This section shows that any rate function, obeying some reasonable restrictions, has a smooth allocation having very efficient dispatchers whose time per-slot is independent of the number of clients, and is essentially constant.

This is achieved by the *binary-intervals* technique which is a variant of the intervals technique established in our earlier paper [22]. Both techniques are based on the shuffle function μ and allocate disjoint sets of slots by allocating disjoint real intervals and applying μ on these intervals. Moreover, both techniques allocate to each client an interval whose length is determined by the client's rate. The main difference between these techniques is that the binary-intervals technique has an additional requirement on the allocated intervals: Each such interval should be a $(* \times *)$-interval. We refer to such an interval as a *binary-interval* and hence the name of the technique. In order to avoid confusion between allocations of slots and allocations of intervals, we henceforth refer to the latter as *assigning*; namely, we allocate sets of slots and assign intervals of real numbers.

The Interval Membership Problem. Our efficient dispatchers are based on efficient solutions to a variant of the following problem. Let Γ be a finite set of clients and let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a Γ-system of disjoint real intervals. In the *S-membership problem* the system S is predefined and the problem is to find, for any given *point* $y \in \mathbb{R}$, the client γ with $y \in S_\gamma$ or to determine that there is no such client.

Our fast dispatchers are based on fast solutions to the S-membership problem for systems of intervals having certain properties. A Γ-system of intervals $S = \{S_\gamma \mid \gamma \in \Gamma\}$ is *solid* if $\cup_{\gamma\in\Gamma} S_\gamma$ is an interval. Such a system is *ordered* if the intervals are arranged in decreasing order of their length; that is, $\|S_\gamma\| < \|S_{\gamma'}\|$ implies that $\inf(S_\gamma) > \inf(S_{\gamma'})$. The following two lemmas are based on elementary data structures and, in order not to interrupt the exposition of the paper, their proofs are deferred to the appendix.

Lemma 8: Let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a solid Γ-system of disjoint intervals s.t. the endpoints of all the S_γ are k-bit rationals; let $q = \max\left\{\|S_\gamma\|/\|S_{\gamma'}\| : \ \gamma, \gamma' \in \Gamma\right\}$ and let $n = |\Gamma|$. Then the S-membership problem can be solved for any point which is a k-bit rational in $O(1)$ time and $O(n \cdot q)$ space on any $\Omega(k)$-bit RAM.

Lemma 9: Let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a solid and ordered Γ-system of disjoint intervals s.t. the endpoints of all the S_γ are k-bit rational; let $q = \max\left\{\|S_\gamma\|/\|S_{\gamma'}\| : \ \gamma, \gamma' \in \Gamma\right\}$ and let $n = |\Gamma|$. Then the S-membership problem can be solved for any point which is a k-bit rational in $O(\log\log q)$ time and $O(n)$ space on any $\Omega(k)$-bit RAM.

The following theorem is among the main results of this paper, and it uses the following terminology (already defined in the introduction). A *k-bit fraction* is a (non-negative) number having a binary expansion with k bits to the right of the binary point; a number is of l *significant-bits* if its binary expansion has an interval of l bits s.t. all the bits outside this interval are 0. In other words, the former number is of the form $j/2^k$ for some $j \in \mathbb{N}$ and the latter number is of the form $i \cdot 2^j$ with $j \in \mathbb{Z}$, $i \in \mathbb{N}$ and $i < 2^l$.

Theorem 2: Let ρ be a rate function over Γ s.t. $\sum_\gamma \rho_\gamma \leq 1$ and all the ρ_γ are k-bit fractions of l significant-bits. Let $q = \max\left\{\rho_\gamma/\rho_{\gamma'} \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is an $(l+1)$-smooth ρ-allocation which has two dispatchers on any $\Omega(k)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space and the second of $O(1)$ time and $O(n \cdot q)$ space.

The proof of Theorem 2 uses the following lemmas and is based on a reduction from the problem of allocating and dispatching a given rate function ρ obeying the premise of Theorem 2 to the problem of producing an appropriate assignment of the unit interval and solving the resulting membership problem.

Lemma 10: Let ρ be a rate function over Γ, $\sum_\gamma \rho_\gamma \leq 1$ and let $\rho_\gamma = h_\gamma/2^{j_\gamma}$ where $h_\gamma, j_\gamma \in \mathbb{N}$. Then:

a. There is a solid Γ-system of disjoint real intervals $S = \{S_\gamma \mid \gamma \in \Gamma\}$ s.t. $0 \in \cup_\gamma S_\gamma$ and for each γ, S_γ is an $(h_\gamma \times j_\gamma)$-interval.

b. Moreover, if for some $l \in \mathbb{N}$, $2^{l-1} \leq h_\gamma < 2^l$ for all γ then there is an ordered Γ-system satisfying statement (a).

Proof: Consider statement (a). We process the clients $\gamma \in \Gamma$ in increasing order of their j_γ, and assign to them adjacent intervals of the appropriate length where the first interval starts at 0. The interval assigned to a client γ is an (h_γ, j_γ)-interval since the length of the previous intervals are all j_γ-bit fractions and the set of the j_γ-bit fractions is closed under addition.

The construction for statement (b) is identical to that for statement (a), except that we process the clients in decreasing order of their rates. We have $\lfloor \log \rho_\gamma \rfloor = \lfloor \log(h_\gamma/2^{j_\gamma}) \rfloor = l-1-j_\gamma$; hence the clients are actually processed in increasing order of their j_γ and the conclusion of statement (a) also holds. $\square$

Lemma 11: Let $k \in \mathbb{N}$ and let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a solid Γ-system of disjoint $(* \times k)$-intervals. Let $q = \max\left\{\|S_\gamma\|/\|S_{\gamma'}\| \mid \gamma, \gamma' \in \Gamma\right\}$, let $n = |\Gamma|$ and let $A = \{\mu(S_\gamma) \mid \gamma \in \Gamma\}$. Then A is a Γ-allocation such that:

a. The allocation A has a dispatcher of $O(1)$ time and $O(n \cdot q)$ space on any $\Omega(k)$-bit RAM.

b. If S is ordered then A has a dispatcher of $O(\log\log q)$ time and $O(n)$ space on any such RAM.

Proof: Consider statement (a). By Lemma 8, the S-membership problem can be solved for any point which is a k-bit rational in $O(1)$ time and $O(n \cdot q)$ space on any $\Omega(k)$-bit RAM. Clearly, each S_γ is a $(* \times k)$-interval. Thus, for any $i \in \mathbb{N}$, $i \in \mu(S_\gamma)$ iff $\nu^k(i) \in S_\gamma$. By Lemma 7, the infinite sequence $\langle \nu^k(0), \nu^k(1), \ldots \rangle$ can be generated in $O(1)$ space and $O(1)$ time per element on any $\Omega(k)$-bit RAM. Hence, each slot can be dispatched in $O(1)$ time and $O(n \cdot q)$ space on any $\Omega(k)$-bit RAM. Statement (b) follows from the same arguments, using Lemma 9 instead of Lemma 8. $\square$

Theorem 2 follows from the fact that any real number $x \in (0,1]$ of l significant-bits equals $h/2^j$ for some $j, h \in \mathbb{N}$ with $2^{l-1} \le h < 2^l$ and from Lemmas 10, 11 and 6(a).

Rational Rates In Theorem 2 the rates are restricted to be binary-fractions having a certain number of significant bits. As stated by the following theorem, this restriction can be lifted if we agree to utilize only 99% of the resource, rather than 100%. (But, of course, the rates still have to be rational numbers and the width of the RAM depends on the complexity of these numbers.)

Theorem 3: Let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \le 0.99$ and all the ρ_γ are k-bit rational. Let $q = \max\left\{\rho_\gamma/\rho_{\gamma'} \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth ρ-allocation which has two dispatchers on any $\Omega(k)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space and the second of $O(1)$ time and $O(n \cdot q)$ space.

We construct the ρ-allocation of Theorem 3 by the following two steps reduction. We slightly increase the rate of each client to a number of 8 significant-bits and solve the resulting allocation problem via Theorem 2. Then we prune the allocated sets of slots to achieve the required rates. The following lemma establishes the first step in this reduction.

Lemma 12: Let $\alpha \in \mathbb{R} \cap (0,1]$. Then there is a number α' of 8 significant-bits s.t. $\alpha \le \alpha' < \alpha/0.99$. Moreover, if $\alpha \ge 2^{-k}$ then α' is a $(k+7)$-bit fraction.

Proof: Let $j \in \mathbb{N}$ be s.t. $2^{-j} \le \alpha < 2^{-j+1}$ and let $\alpha' = \left\lceil \alpha \cdot 2^{j+7} \right\rceil / 2^{j+7}$. Since $128 \le \left\lceil \alpha \cdot 2^{j+7} \right\rceil \le 256$ and since the set of numbers of l significant-bits is closed under multiplication by 2^i for any $i \in \mathbb{Z}$, α' is of 8 significant-bits. We have:

$$\begin{array}{lllll} \alpha & \le & \alpha' & < & \alpha + 1/2^{j+7} \quad \text{since } \alpha' = \left\lceil \alpha \cdot 2^{j+7} \right\rceil / 2^{j+7} \\ & & & \le & \alpha(1+1/2^7) \quad \text{since } 2^{-j} \le \alpha \\ & & & < & \alpha/0.99 \quad \text{since } 1+1/2^7 < 1/0.99 \end{array}$$

Clearly, our α' is a $(j+7)$-bit fraction; hence, if $\alpha \ge 2^{-k}$ then $k \ge j$ and α' is a $(k+7)$-bit fraction, as required. $\square$

The following lemma summarizes the second step of the above reduction in which the allocated sets are pruned to have the required rates.

Lemma 13: Let $A = \{A_\gamma \mid \gamma \in \Gamma\}$ be a Γ-allocation having a dispatcher of t time and s space on a certain RAM; let $B = \{B_\gamma \subseteq \mathbb{N} \mid \gamma \in \Gamma\}$ be a Γ-system and let each B_γ have a resolver of $O(1)$ time and $O(1)$ space on the above RAM. Then the Γ-system $C = \{A_\gamma \circ B_\gamma \mid \gamma \in \Gamma\}$ is a Γ-allocation having a dispatcher of $t + O(1)$ time and $s + |\Gamma|$ space on the above RAM.

Proof: Since $A_\gamma \circ B_\gamma \subseteq A_\gamma$, C is a Γ-allocation. Our dispatcher of C dispatches each slot i in two steps. First it dispatches slot i as per the allocation A. If i is not allocated in A then the same holds for C. Otherwise, let γ be the client owning this slot in A. Our dispatcher invokes the resolver of B_γ to resolve the next slot of this resolver. According to that resolution our dispatcher either dispatches slot i to γ or determines that the slot is not allocated. $\square$

Theorem 3 follows from Theorem 2 and Lemmas 12, 2, 3, 5(a) and 13.

4 Multiple Resources

A natural extension of our work concerns scheduling of m identical resources where the sum of the rates of the clients is (of course) at most m and the rate of each client is at most one. If it is permitted to serve a single client by several resources in one slot then it is easy to transform any scheduling technique of a single resource into a scheduling technique of m identical resources, as follows. Split each slot into m consecutive sub-slots; schedule the resulting infinite sequence of sub-slots by the given technique on a single resource; replace the service provided by this resource during m consecutive sub-slots with the equivalent service provided by m resources in a single slot.

However, in some applications (e.g., CPU scheduling) a client can be served concurrently by at most one resource, and it was widely believed that this restriction "adds a surprising amount of difficulty" (to the scheduling problem) [23, 8]. In this section we show that our binary-intervals technique has a straightforward extension to the case of multiple resources even under the restriction of no concurrent service.

Our extensions are summarized in the following two theorems, which are generalization of Theorems 2 and 3, and use the following terminology in the context of m identical resources. For a set of clients Γ, a (Γ, m)-*allocation* is a Γ-system $\{A_\gamma \subseteq \mathbb{N} \mid \gamma \in \Gamma\}$ s.t. for any slot $i \in \mathbb{N}$, the set of clients served in this slot, $\{\gamma \mid i \in A_\gamma\}$, has at most m clients. Note that the allocation specifies which clients are served in each slot but does not specify which client each resource serves. For a rate function ρ over Γ, a (ρ, m)-allocation is a (Γ, m)-allocation $\{A_\gamma \mid \gamma \in \Gamma\}$ with $\text{rate}(A_\gamma) = \rho_\gamma$. A *dispatcher* of such an allocation is an algorithm that computes, for each slot i, the set of clients that are served in this slot (but does not specify which client each resource serves).

Theorem 4: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \le m$, each $\rho_\gamma \le 1$ and all the ρ_γ are k-bit fraction of l significant-bits. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is an $(l+1)$-smooth (ρ, m)-allocation which has two dispatchers on any $\Omega(k + \log n)$-bit RAM: the first of $O(m \log\log q)$ time and $O(n)$ space and the second of $O(m)$ time and $O(n \cdot q)$ space.

Theorem 5: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \le 0.99 \cdot m$, each $\rho_\gamma \le 1$ and all the ρ_γ are k-bit rationals. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth (ρ, m)-allocation which has two dispatchers on any $\Omega(k + \log n)$-bit RAM: the first of $O(m \log\log q)$ time and $O(n)$ space, and the second of $O(m)$ time and $O(n \cdot q)$ space.

The allocation constructed below for Theorems 4 and 5 has an additional interesting property — it can be realized in such a way that each client is served by at most two resources.

Recall that the binary-intervals technique (as well as the intervals technique) is based on the shuffle μ which is defined only on binary-fractions in the unit interval. The extension of our binary-intervals technique to several identical resources is based on the function $\bar{\mu}$ which is the following straightforward extension of μ to all positive binary-fractions: $\overline{\mu}(x) \triangleq \mu(x \bmod 1)$. The following lemma follows from the fact that μ is a shuffle (and hence one-to-one) and is onto the natural numbers; the specific definition of μ is irrelevant as far as this lemma is concerned.

Lemma 14:

a. For any interval $X \subset \mathbb{R}^+$ with $\|X\| \le 1$, $\text{rate}(\overline{\mu}(X)) = \|X\|$.

b. Let $l \in \mathbb{N}$ and $X \subset \mathbb{R}^+$ be an interval which is closed in one side and open in the other with $\|X\| = l$. Then $|\{x \in X \mid \overline{\mu}(x) = i\}| = l$ for any $i \in \mathbb{N}$.

In this section we adjust the notion of $(h \times j)$-interval and its derivatives to fit the context of multiple resources by lifting the requirement that these intervals are sub-intervals of the unit interval. Most of our Lemmas hold for this new definition, when μ is replaced with $\bar{\mu}$. In particular, we use the adjusted variants of the following two lemmas: Lemma 6 (with no additional adjustments) and Lemma 10 with the omission of the requirement of $\sum_\gamma \rho_\gamma \leq 1$.

In the following we present the extension of the binary-intervals technique to multiple resources in the context of Theorem 4; the deduction from this theorem to Theorem 5 is the same as in the single resource case.

Given a rate function ρ over Γ as per Theorem 4, we construct a Γ-system $\{S_\gamma \mid \gamma \in \Gamma\}$ of disjoint sub-intervals of the real interval $[0, m)$ satisfying (the new version of) Lemma 10. Let $A_\gamma \triangleq \overline{\mu}(S_\gamma)$ and let $A = \{A_\gamma \mid \gamma \in \Gamma\}$. By Lemma 14, A is an (ρ, m)-allocation and, moreover, by Lemma 6(a), this allocation is $(l+1)$-smooth.

It remains to show that A has efficient dispatchers as per Theorem 4. To dispatch slot i, one needs to solve m instances of the S-membership problem for the points $\mu^{-1}(i), \mu^{-1}(i)+1, \cdots, \mu^{-1}(i)+m-1$. Our dispatchers take advantage of the fact that the endpoints of all the S_γ are k-bit fractions and solves the S-membership problem for the points $\nu^k(i), \nu^k(i)+1, \ldots, \nu^k(i)+m-1$. By Lemma 7, $\nu^k(i)$ can be generated in $O(1)$ space and $O(1)$ time on any $\Omega(k)$-bit RAM. Assume, without loss of generality, that $m \leq n$; hence, the endpoints of all the S_γ are $(k + \log n)$-bit rationals; by Lemma 8, the slot can be dispatched in $O(m)$ time and $O(n \cdot q)$ space on any $\Omega(k+\log n)$-bit RAM and by Lemma 9, the slot can be dispatched in $O(m \log\log q)$ time and $O(n)$ space on any such RAM.

4.1 Independent Dispatching

In some applications (e.g., CPU scheduling) each resource has its own computational capabilities. In this case it is highly desirable that each resource independently dispatches its own slots; that is, in each slot the resource computes what client to serve. As stated formally by the following two theorems, our scheduling technique provide independent dispatchers whose time per-slot does not depend on the number of clients or on the number of resources, and is either constant or essentially constant. Moreover, the service provided by each single resource is 10-smooth.

Recall that a sequence $A^1, \cdots, A^m$ of Γ-allocations *realizes* a (Γ, m)-allocation A if when each resource j serves the clients as per the allocation A^j then the combined service is as per the (Γ, m)-allocation A; namely, no slot is assigned to the same client by two distinct A^j allocations and in each time-slot i the set $\left\{\gamma \mid \exists j :\ i \in A^j_\gamma\right\}$ of clients served by the combined allocation in this time-slot equals to the set of clients served by the (Γ, m)-allocation A in that time-slot. As usual, the first of the following two theorems utilizes 100% of the resource but restricts the rates to be binary fractions having a certain number of significant bits; the second theorem lifts this restriction but utilizes only 99% of the resource.

Theorem 6: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \leq m$, each $\rho_\gamma \leq 1$ and all the ρ_γ are k-bit fraction of l significant-bits. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is an $(l+1)$-smooth (ρ, m)-allocation which is realized by a sequence $A^1, A^2, \cdots A^m$ of Γ-allocations s.t. each of the A^j is $(l+1)$-smooth and has two dispatchers on any $\Omega(k+\log n)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space, and the second of $O(1)$ time and $O(n \cdot q)$ space.

Proof: The construction for this theorem follows closely that of Theorem 4. The assignment of intervals to clients $S = \{S_\gamma \mid \gamma \in \Gamma\}$, the resulting (ρ, m)-allocation $A = \{A_\gamma \mid \gamma \in \Gamma\}$, and the databases used to solve the S-membership problem are exactly the same in both theorems; thus, the (ρ, m)-allocation A is $(l+1)$-smooth. In the case of Theorem 4, a slot is dispatched by solving m instances of the S-membership problem. In the current case, the j-th resource dispatches slot i by solving the j-th instance; that is, at time i the j-th resources solves the membership problem of $\nu^k(i)+j-1$. Thus, resource j is associated with interval $[j-1, j)$ and solves the S-membership problem exclusively for points in this interval. This dispatching produces a Γ-allocation of the slots of the j-th resource denoted by $A^j = \left\{A^j_\gamma \mid \gamma \in \Gamma\right\}$; by Lemma 6(a), A^j is $(l+1)$-smooth and by Lemma 11, A^j has dispatchers of the required complexity. By our construction and by Lemma 14, A is realized by the A^j allocations. □

Theorem 7: Let $m \in \mathbb{N}$ and let ρ be a rate function over Γ s.t. $\sum_{\gamma\in\Gamma} \rho_\gamma \le 0.99 \cdot m$, each $\rho_\gamma \le 1$ and all the ρ_γ are k-bit rationals. Let $q = \max\left\{\rho_\gamma/\rho'_\gamma \mid \gamma, \gamma' \in \Gamma\right\}$ and $n = |\Gamma|$. Then there is a 10-smooth (ρ, m)-allocation which is realized by a sequence $A^1, A^2, \cdots A^m$ of Γ-allocations s.t. each of the A^j is 10-smooth and has two dispatchers on any $\Omega(k + \log n)$-bit RAM: the first of $O(\log\log q)$ time and $O(n)$ space, and the second of $O(1)$ time and $O(n \cdot q)$ space.

Proof: As in the construction of Theorems 3 and 5, we construct the required (ρ, m)-allocation by the following two steps reduction. We slightly increase the rate of each client to a number of 8 significant-bits and solve the resulting problem using Theorem 6. Then we prune the allocated sets of slots so that they have the required rate. This pruning has to be done carefully due to the following problem which is expressed in the terminology of Theorem 6. An interval S_γ may intersect two consecutive intervals $[j, j+1)$ and $[j+1, j+2)$. If each of the two resources in question prunes its allocation of γ then the total allocation of γ may fail to be 10-smooth. To overcome this problem, the pruning should be done by only one of these resources. This is possible since we need to prune out much less than half of the current allocation.

We now show that the resulting total allocation A_γ of such a γ is 10-smooth. For $i = 1, 2$, let $Y_i = S_\gamma \cap [j+i-1, j+i)$. For any (1×0)-interval Z and $(1\times *)$-interval X, either $X \subset Z$ or $X \cap Z = \emptyset$. This and Lemma 6(b) implies that $\bar{\mu}(Y_i)$ is Δ_i-smooth for some Δ_1 and Δ_2 with $\Delta_1 + \Delta_2 = 9$. Exactly one of these sets, say $\bar{\mu}(Y_1)$, is pruned. By Lemma 5(a), the pruned set is $(\Delta_1 + 1)$-smooth and by Lemma 1(a), the resulting A_γ is 10-smooth. $\square$

5 Scheduling of Communication Networks

This section demonstrates that the concept of smoothness is very attractive for scheduling the links of connection-oriented packet switching networks, and in particular of these networks that transport continuous-media (video, audio, etc.) of real-time nature. The section also demonstrates that, in this application, the difference between a 2-smooth schedule (as in [8]) and a 10-smooth schedule (as in Section 3) is not essential.

We assume the standard model of packet switching network [19] with a small modification, as described in Subsection 1.4. In our case the network is used by *connections* (a.k.a. *sessions*) where each connection is a sequence of packets (a *packet-stream*) that traverses a predefined path from its *source* to its *destination*; the packets of a packet-stream follow each other (and never get lost); in each time-slot the connection initiates at most one packet; and each connection γ has a *rate* ρ_γ which constrains its bandwidth as follows. In any k consecutive slots the connection initiates at most $\rho_\gamma \cdot k + \Delta'$ packets, where Δ' is some global constant. Many works assume such a constraint (e.g., [28, 14, 15, 25, 16, 10, 6, 17]).

Our networking scheme is very simple due to the following features. Each link is pre-allocated to the connections using it and this allocation is Δ-smooth for a small Δ. (It is assumed that $\Delta < \Delta'$). Moreover, the allocation of each link is done in isolation, without regard to the other links. This enables very fast dispatching via the techniques of Section 3. Moreover, once the allocation of slots has been done the connections do not compete with each other and therefore the analysis of the traffic is very simple.

In the context of network traffic it is convenient to seperate the two ingredients of the concept of smoothness as follows. A set $A \subset \mathbb{N}$ is $(<, \rho, \Delta)$-*bounded* if $|A \cap I| < |I| \cdot \rho + \Delta$ for any interval I of natural numbers; it is $(>, \rho, \Delta)$-*bounded* if $|A \cap I| > |I| \cdot \rho - \Delta$ for any such interval I. By definition, a set is (ρ, Δ)-smooth iff it is both $(<, \rho, \Delta)$-bounded and $(>, \rho, \Delta)$-bounded. The following lemma is straightforward.

Lemma 15: Let A_1 be $(<, \rho_1, \Delta_1)$-bounded, let A_2 be $(>, \rho_2, \Delta_2)$-bounded, and let I be an interval of $\mathbb{N}$. Then $|(A_1 \cap I)| - |(A_2 \cap I)| < (\rho_1 - \rho_2)|I| + \Delta_1 + \Delta_2$.

An *interface* of a communication network is a (virtual) boundary within the network which is crossed in any time-slot by at most one packet; an example is the head of a link. Given an interface x and a packet-stream P that crosses x, we denote by P_x the set of slots in which packets of P cross x, and use the phrase 'the packet-stream P at x' to refer to this set. For example, we say that the packet-stream P is (ρ, Δ)-smooth at x to denote that P_x is (ρ, Δ)-smooth.

The main components of our networking scheme are stream-shapers of three types. A *stream-shaper* is a (virtual) component that shapes a single packet-stream. It receives the packet-stream through its entry interface and transmits it via its exit interface so that the resulting stream has a certain desired form. A stream-shaper preserves, of course, the order of the packets and may receive a packet and transmit it during the same slot. Our first stream-shaper is associated with a predefined set $B \subset \mathbb{N}$ and is called a B-*doorman*. It shapes its packet-stream P so that P at the exit of the shaper is a subset of B; moreover, this stream-shaper is a greedy one: it releases a packet in every slot of B unless it has no packet. A (ρ, Δ)-*doorman* is a B-*doorman* for some (ρ, Δ)-smooth set B.

Lemma 16: Assume that a packet-stream P enters a (ρ, Δ_1)-doorman D and that P is $(<, \rho, \Delta_2)$-bounded at the entry of D. Then:

a. At any instant, the doorman stores at most $\Delta_1 + \Delta_2$ packets of P.

b. The doorman delays any packet of P by at most $(\Delta_1 + \Delta_2)/\rho$ slots.

Proof: Let B be the set associated with the doorman, let x and y be the entry and exit of D, and let $b(t)$ be the number of packets the doorman has at the beginning of slot t. For statement (a), consider a slot t and let $t' \triangleq \max\{\tau \leq t : \ b(\tau) = 0\}$; as $b(0) = 0$, the above set is not empty. Let $I = [t', t)$. By Lemma 15, we have:

$$b(t) = |P_x \cap I| - |P_y \cap I| < \Delta_1 + \Delta_2$$

For statement (b), let a packet p enter the doorman during slot t_1 and exit it during slot t_2. Let $t' \triangleq \max\{\tau \leq t_1 : \ b(\tau) = 0\}$, let $I_1 = [t', t_1]$ and let $I_2 = [t', t_2]$. The doorman delays the packet p by $|I_2| - |I_1|$. We have:

$$|I_1| \cdot \rho + \Delta_2 > |P_x \cap I_1| = |P_y \cap I_2| = |B \cap I_2| > |I_2| \cdot \rho - \Delta_1$$

□

The other two stream-shapers we use are related to jitter. The (end-to-end) *jitter* [28] of a connection is the maximum difference between the (end-to-end) delays of any two packets of that connection. Clearly, it is desirable (or even mandatory) to transport continuous media with zero jitter, and this can be easily achieved when an upper bound on the delay is given [27]. In our scheme, we eliminate a jitter (not necessarily an end-to-end one) by a pair of stream-shapers, a *timestamp* and an *equalizer*. Such a pair equalizes the delay of the packets of a certain connection from one interface of the network to another, as follows. The timestamp is located at the first interface and stamps the packets (of the connection in question) with the current time, without introducing any delay. The equalizer is located at the second interface and is given a bound on the delay from the first to the second interface. It equalizes the delay of the packets, from the first interface to the exit of the equalizer, so that all these delays are equal to the above given bound.

Lemma 17: Let a packet-stream γ pass through a pair of a timestamp T and an equalizer E. Assume that γ is $(<, \rho, \Delta)$-bounded at the entry of E, the delay of packets from T to E is between t_1 and t_2 and that t_2 is the bound given to E. Then E stores, at most, $(t_2 - t_1) \cdot \rho + \Delta$ packets.

Proof: By definition, E delays a packet for at most $t_2 - t_1$ slots. Hence, packets queued at the equalizer at the beginning of a slot have entered it during the previous $t_2 - t_1$ slots; since the packet-stream is $(<, \rho_\gamma, \Delta)$-bounded at the entry of E, the number of such packets is less than $(t_2 - t_1) \cdot \rho + \Delta$.
□

In what follows we describe three networking schemes. In all of them each link is allocated, in isolation, to the connection using it and this allocation is Δ-smooth for a small Δ and has a fast dispatcher. Such an allocation is presented in Section 3; however, any allocation having the above attributes will do. Once this allocation has been done, the connections do not compete with each other any more. Therefore, our schemes handle each connection in isolation and we henceforth present how each of these schemes handles a single connection γ. Due to the simplicity of this approach, each scheme can be completely specified by a diagram, as shown in Figures 1, 2, and 3.

Our first networking scheme, shown in Figure 1, is a semi-greedy one as follows. Let L_i be the i-th link of the path of γ and let B_i be the set of slots of L_i allocated to γ. Connection γ uses every slot of B_i unless it has no available packet. (Of course, γ does not use other slots of L_i, even if they are idle, hence the term *semi-greedy*.) To this end, γ enters L_i via a B_i-doorman, denoted D_i in Figure 1. To eliminate the end-to-end jitter, we use an end-to-end pair, T_1 and E^{T_1}, of a timestamp and an equalizer.

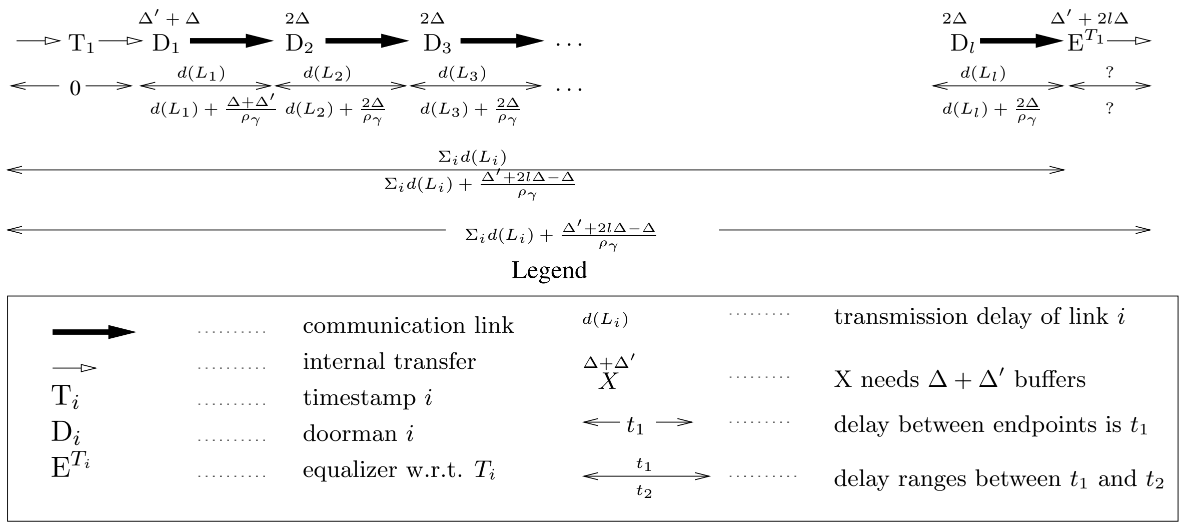

Figure 1: Connection γ under Scheme 1

An important aspect of a networking scheme is its buffering requirements. Under our schemes the buffers are associated with the stream-shapers. As illustrated in Figure 1, connection γ needs $\Delta + \Delta'$, 2Δ and $\Delta' + 2l\Delta$ buffers for the first doorman, for any other doorman, and for the equalizer, respectively, when l is the length of the connection's path. The first two bounds follow from Lemma 16(a) and the last bound is discussed shortly.

Another aspect of a networking scheme, which is particularly important for real-time communication, is the end-to-end delay of the connections. (Note that, once the jitter is eliminated, all packets of γ experience the same delay). As illustrated in Figure 1, the delay of a packet from the entry of doorman D_i to the exit of the following link L_i is at least the *transmission delay* of the link, $d(L_i)$, and at most $d(L_i) + (\Delta + \Delta')/\rho_\gamma$ for $i = 1$ and $d(L_i) + 2\Delta/\rho_\gamma$ for $i > 1$. These two bounds follows from Lemma 16(b). Thus, the delay of a packet, from the origin up to the entry of the equalizer, is between $\sum_{i=1}^{l} d(L_i)$ and $(\Delta' + 2l\Delta - \Delta)/\rho_\gamma + \sum_{i=1}^{l} d(L_i)$, when l is the length of its path. These last two numbers, the fact that the packet-stream at the exit of any link is $(<, \rho_\gamma, \Delta)$-bounded and Lemma 17 establish the above bound of $\Delta' + 2l\Delta$ on the buffering requirement of the equalizer. Note that the simplicity of the above analysis is due to the fact that our scheme handles each connection in isolation, without regard to the other connections.

Another important aspect of a networking scheme is its *capacity*. An efficient networking scheme should serve concurrently a large set of connections, and therefore the restrictions it imposes on those connection should be as weak as possible. Our scheme, when using the scheduling techniques of Section 3, imposes only the following restrictions on the connections: The packet-stream of each connection γ is $(<, \rho_\gamma, \Delta')$-bounded at the origin (where Δ' is an arbitrary global constant) and the sum of the rates of the connections using each link is at most 0.99. We say that such a scheme has a *capacity* of 0.99, and all our schemes possess this high capacity.

It is highly desirable [28] that the online algorithms controlling the links are very fast. In a naive implementation of our scheme, a link used by n connections actually has n doormen. Hence, the selection of a packet to be transmitted in a slot takes $\Omega(n)$ time even if each doorman need just $O(1)$ time per slot. However, the doorman concept is useful for analyzing the behavior of the network but not for an efficient implementation. In such an implementation, a doorman is not an active entity but a passive queue of packets and the active elements are only the links. In each time-slot at most one packet is added to the queue of a doorman and this can be done by the link submitting the packet. In addition, at most one packet enters a link in each time-slot; the selection of the connection which uses the current slot is performed by the centralized dispatcher of the link. Once this selection is made, the link extracts and transmits the first packet from the appropriate queue (if the queue is not empty).

Thus, the computational complexity of the scheduling decision performed by each link is derived from the complexity of the dispatcher associated with the link. Assume that we use the allocation and dispatching of Section 3, let n be the number of clients using a certain link and let q be the maximum ratio of the rates of two clients using this link. In many applications the value of $(\log\log q)$ is actually a small constant; thus, by Theorem 3, the complexity of the dispatcher of the above link is $O(n)$ space and $O(1)$ time per-slot.

As said, our first scheme is semi-greedy — a link is left idle in a slot owned by γ only if γ has no packet at the corresponding switch. Due to this greediness, the buffering requirement of the equalizer is exceedingly large and depends on the length of the path. This weakness is overcome by our second scheme which is not semi-greedy and, instead of eliminating a large jitter only once at the destination, it eliminates small jitters at each intermediate node. (This idea of piece-wise elimination of the jitter is not new [28]). To this end, an additional timestamp T_2 is inserted after the first doorman and equalizers w.r.t. this timestamp are inserted at the exit of any link except of the first one; see Figure 2.

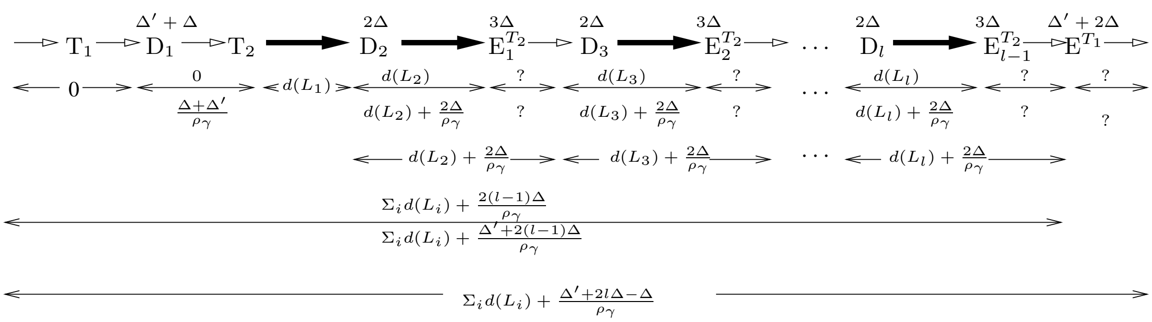

Figure 2: Connection γ under Scheme 2

As in the previous scheme, the bounds on the delays, shown in Figure 2, follows from Lemma 16(b) and from the fact that the packet-stream at the entry of each doorman, except for the first one, are $(<, \rho_\gamma, \Delta)$-bounded. Similarly, the new buffering requirements follow from Lemmas 16(a) and 17. We set the new equalizers to work with T_2 rather than with T_1 since we assumed that $\Delta < \Delta'$ and therefore the other setup would increase the buffering requirements and the end-to-end delay.

In an efficient implementation of this scheme, a link performs, not only the functionality of the doormen, but also the functionality of the equalizers. That is, in each slot a link first computes the connection owning this slot and then examines the first packet of the relevant queue (if this queue is not empty) and checks its time-stamp; if the time-stamp is too recent then the packet remains in the queue and the slot is left idle. Thus, the computational complexity of each link is equal, up to a constant factor, to that of the first scheme.

Our third scheme, shown in Figure 3, reduces the end-to-end delay and the buffer requirements of the system but requires more computation power In this scheme the packet-stream is $(<, \rho_\gamma, 1)$-bounded (rather than $(<, \rho_\gamma, \Delta)$-bounded) at the entry of each doorman. To this end, a new $(\rho_\gamma, 1)$-doorman D_0 is inserted after T_1, the timestamp T_2 is moved to the exit of this doorman and an additional equalizer w.r.t. T_2 is inserted at the exit of the first link L_1; see Figure 3. Under this setup, the packet-stream is $(<, \rho_\gamma, 1)$-bounded at T_2 and therefore it is also $(<, \rho_\gamma, 1)$-bounded at the exit of each intermediate equalizer. As shown in Figure 3, this improves the end-to-end delay and the buffering requirements. However, the functionality of the new doormen can not be performed by the links and these doormen requires additional computational capabilities.

Acknowledgments

We wish to thank Adi Rosen for helpful discussion and references.

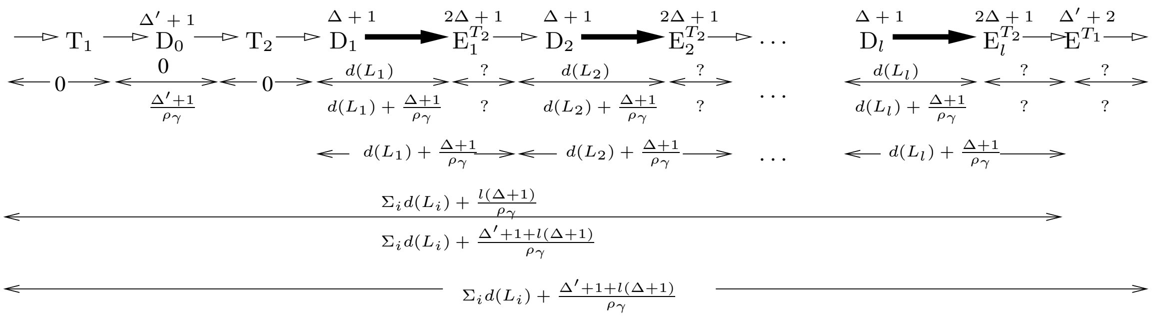

Figure 3: Connection γ under Scheme 3

References

[1] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 101–108, 2003.

[2] J. Anderson and A. Srinivasan. Early-release fair scheduling. In *Proc. of the 12th Euromicro Conference on Real-Time Systems*, pages 35–43, 2000.

[3] J. Anderson and A. Srinivasan. Pfair scheduling: Beyond periodic task systems. In *Proc. of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306, 2000.

[4] J. H. Anderson and A. Srinivasan. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Computer and System Sciences*, 68:157–204, 2004.

[5] B. Andersson, S. K. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. Technical Report TR01-016, Department of Computer Science, University of North Carolina - Chapel Hill, 2001.

[6] M. Andrews, A. Fernandez, M. Harchol-Balter, F. T. Leighton, and L. Zhang. General dynamic routing with per-packet delay guarantees of O(distance + 1 / session rate). *SIAM Journal on Computing*, 30(5):1594–1623, 2000. (Extended abstract was presented at the 38th Annual Symposium on Foundations of Computer Science (FOCS) 1997.).

[7] A. Bar-Noy, A. Mayer, B. Schieber, and M. Sudan. Guaranteeing fair service to persistent dependent tasks. *SIAM J. Of Computing*, 27(4):1168–1189, 1998.

[8] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: a notion of fairness in resource allocation. *Algoritmica*, 15(6):600–625, 1996. (Extended abstract was presented at The 25th Annual ACM Symposium on the Theory of Computing. May 1993.).

[9] S. K. Baruah, J. Gehrke, and G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288. IEEE Computer Society Press, 1995.

[10] J. C. R. Bennet and H. Zhang. $\mathrm{WF}^2\mathrm{Q}$: Worst-case Fair Queueing. In *the Fifteenth INFOCOM*, pages 120–128. IEEE, 1996.

[11] A. Chandra, M. Adler, P. Goyal, and P. Shenoy. Surplus fair scheduling: A Proportional-Share CPU scheduling algorithm for symmetric multiprocessors. In *Proceedings of the USENIX 4th Symposium on Operating System Design and Implementation*, pages 45–58, 2000.

[12] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proceedings of the 7th IEEE Real-Time Technology and Applications Symposium*, pages 3–14, 2001.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press and McGraw-Hill Book Company, second edition, 2001.

[14] R. L. Cruz. A calculus for network delay, part i: Network elements in isolation. *In IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[15] R. L. Cruz. A calculus for network delay, part ii: Network analysis. *In IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

[16] A. Demers, S. Keshave, and S. Shenkar. Analysis and simulation of a fair queueing algorithm. *Journal of Internet-working Research & Experience*, pages 3–12, 1990.

[17] S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *the Thirteenth INFOCOM*, 1994.

[18] P. Holman and J. Anderson. Guaranteeing pfair supertasks by reweighting. In *Proceedings of the 22nd IEEE Real-time Systems Symposium*, pages 203–212, 2001.

[19] F. T. Leighton. *Introdution to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann.

[20] A. J. Lincoln, S. Even, and M. Cohn. Smooth pulse sequences. In *Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems*, pages 350–354, 1969.

[21] A. Litman and S. Moran-Schein. On distributed smooth scheduling. Technical Report CS-2005-03, Department of Computer Science, Technion - Israel Institute of Technology, 2005. Available at: www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2005/CS/CS-2005-03.

[22] A. Litman and S. Moran-Schein. On smooth sets of integers. Technical Report CS-2005-02, Department of Computer Science, Technion - Israel Institute of Technology, 2005. Available at: www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2005/CS/CS-2005-02.

[23] C. L. Liu. Scheduling algorithms for multiprocessors in hard-real-time enviroment. *JPL space program summary* 37-60, vol. II, Propulsion Lab., Calif. Inst. of Tech., Pasadena, CA, pages 28–37, 1969.

[24] M. Moir and S. Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *The 20th IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

[25] A. K. Perekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services network-the single node case. *ACM/IEEE Trans. on Networking*, 1(3):344–357, 1992.

[26] R. Tijdeman. The chairman assignment problem. *Discrete Mathematics*, 32:323–330, 1980.

[27] D. C. Verma, H. Zhang, and D. Ferrari. Delay jitter control for real-time communication in a packet switching network. In *proceeding of TriComm*, pages 35–43, 1991.

[28] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proc. IEEE*, volume 83, pages 1374–1396, 1995.

Appendix

Lemma 8: Let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a solid Γ-system of disjoint intervals s.t. the endpoints of all the S_γ are k-bit rationals; let $q = \max\left\{\|S_\gamma\|/\|S_{\gamma'}\| :\ \gamma, \gamma' \in \Gamma\right\}$ and let $n = |\Gamma|$. Then the S-membership problem can be solved for any point which is a k-bit rational in $O(1)$ time and $O(n \cdot q)$ space on any $\Omega(k)$-bit RAM.

Proof: For the sake of simplicity, assume that all intervals S_γ are closed at the bottom and open at the top. Let $U \triangleq \cup_\gamma S_\gamma$ and $r_{\min} \triangleq \min\{\|S_\gamma\| \mid \gamma \in \Gamma\}$. Consider the arithmetic sequence $W =$

$\langle w_0, w_1, \cdots \rangle$ in which $w_0 = \min(U)$ and $w_{i+1} - w_i = r_{\min}$. We construct a linear array with $\lceil \|U\|/r_{\min} \rceil$ entries in which the i-th entry contains enough information to determine, for any $y \in [w_i, w_{i+1}) \cap U$, the client γ s.t. $y \in S_\gamma$. Namely, the i-th entry contains a real number z and two clients $\gamma, \gamma' \in \Gamma$ s.t. $[w_i, z) \subset S_\gamma$ and $[z, w_{i+1}) \cap U \subset S_{\gamma'}$. (However, γ and γ' can be equal.) This is possible since each interval $[w_i, w_{i+1})$ intersects at most two intervals of S.

Given a point y which is a k-bit rational, the algorithm determines whether $y \in U$. If so, the algorithm computes the index i s.t. $y \in [w_i, w_{i+1})$ and, using the data in this entry, finds the required γ. Clearly, this can be done in $O(1)$ time on any $\Omega(k)$-bit RAM. Consider the space requirements. Clearly, $\|U\| \le r_{\min} \cdot q \cdot n$. Thus, the array has at most $\lceil n \cdot q \rceil$ entries. Moreover, the data of each entry can be encoded in $O(1)$ space, implying that the array consumes $O(n \cdot q)$ space. $\square$

Lemma 9: Let $S = \{S_\gamma \mid \gamma \in \Gamma\}$ be a solid and ordered Γ-system of disjoint intervals s.t. the endpoints of all the S_γ are k-bit rational; let $q = \max\left\{\|S_\gamma\|/\|S_{\gamma'}\| : \ \gamma, \gamma' \in \Gamma\right\}$ and let $n = |\Gamma|$. Then the S-membership problem can be solved for any point which is a k-bit rational in $O(\log\log q)$ time and $O(n)$ space on any $\Omega(k)$-bit RAM.

Proof: For each $i \in \mathbb{Z}$, define $\Gamma_i \triangleq \left\{\gamma \mid 2^{i-1} \le \|S_\gamma\| < 2^i\right\}$ and $X_i \triangleq \cup_{\gamma \in \Gamma_i} S_\gamma$. Since S is ordered, each X_i is an interval; let each non-empty X_i be called a *cluster*. Given an instance $y \in \mathbb{R}$ of the S-membership problem, our algorithm determines its membership in two steps. In the first step it determines the cluster to which y belongs (if such a cluster exists) and in the second step it determines the actual interval S_γ to which y belongs.

By Lemma 8, the second step is of $O(1)$ time and $O(n)$ space. Consider the first step. The algorithm uses a binary search tree that has been constructed recursively as follows. The root contains a real number $z \in [0, 1)$ which is a 'median' of the clusters; that is, the number of clusters to its right and the number of clusters to its left differ by at most one; each internal node is constructed analogously w.r.t. to the relevant clusters. Each leaf contains the endpoints of a single cluster. The number of clusters is no more than n and no more than $(\log q + O(1))$. Thus, the depth of the tree is $O(\log\log q)$ and the number of nodes is $O(n)$. Since the endpoints of each S_γ are k-bit rationals, the space requirements of each internal node and each leaf is $O(1)$ on any $\Omega(k)$-bit RAM. Putting everything together, the algorithm is of $O(\log\log q)$ time and $O(n)$ space on any $\Omega(k)$-bit RAM. $\square$