Design and analysis of a class-aware recursive loop scheduler for class-based scheduling

Design and Analysis of a Class-aware Recursive Loop Scheduler for Class-based Scheduling

Raphael Rom and Moshe Sidi
Department of Electrical Engineering
Technion, Israel Institute of Technology
Technion City 32000, Israel

Hwee Pink Tan*
EURANDOM, Eindhoven University of Technology,
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

ABSTRACT

In this paper, we consider the problem of devising a loop scheduler that allocates slots to users according to their relative weights as smoothly as possible. Instead of the existing notion of smoothness based on balancedness, we propose a variance-based metric which is more intuitive and easier to compute.

We propose a recursive loop scheduler for a class-based scheduling scenario based on an optimal Weighted Round Robin scheduler. We show that it achieves very good allocation smoothness with almost no degradation in intra-class fairness. In addition, we also demonstrate the equivalence between our proposed metric and the balancedness-based metric.

Keywords: Loop Scheduler, Smoothness, Recursive, Class-aware scheduling

1 Introduction

Consider a scenario where an indivisible resource (e.g., time slot) is to be shared amongst K users by means of time multiplexing. The components that are relevant to this resource-allocation problem are the *service* discipline or *scheduler*, the *users* and the *performance* metrics.

Users can be characterized in terms of their type (jobs vs streams) and arrival characteristics (time and size). The scheduler determines how much resource, which user and when to allocate the resource. The scheduling mechanism may depend on the user characteristics (online) or offline (independent of arrival characteristics). Finally, such a system is usually evaluated in terms of its *efficiency* (e.g., mean waiting time in the system) and *fairness* (e.g., how fairly each user is treated). While efficiency has been extensively studied and is well-understood, there is still no commonly agreed upon theoretical yardsticks for measuring fairness in resource-allocation systems.

A common notion of fairness is the following: an allocation is *fair* if at every instant of time, each user is receiving its fair share (instantaneous fairness). Assuming that each user has equal *rights* or priority to the resource, a hypothetical service discipline that is fair is a bit-by-bit round-robin (BR) discipline, since at every instant in time, each user is receiving its fair share. For users that can be characterized as *jobs* (e.g., in a supermarket set-up), the Processor-sharing model (introduced in [1] and generalized in [2]) is commonly used to evaluate the efficiency of the BR discipline as the bit size tends to zero. Recently, a resource-allocation queueing fairness measure (RAQFM) has been proposed in [3] that accounts for both seniority differences (times of arrival) and service-time differences (job sizes). The authors also analyzed the fairness performance of four basic queueing systems, where the results shed some light on job-level unfairness as a function of the service discipline.

On the other hand, if each user corresponds to a *stream* of packets (e.g., in communication systems), the notion of fairness is modified to take into account the idle periods between packet arrivals to a user as follows: an allocation is *fair* if for an interval, τ, over which each user is backlogged, it receives its fair share (fairness over τ). Since sending packets in a bit-by-bit fashion is unrealistic, packet-by-packet transmission schemes have been proposed to emulate the BR scheme, most of which are variants of the Weighted-Fair Queueing (WFQ) discipline [4]. This was extended in [5] with the Generalized Processor Sharing (GPS) discipline (and corresponding

*Corresponding author. E-mail address: tan@eurandom.tue.nl. This work was carried out when the author was a PhD candidate in the Department of Electrical Engineering, Technion, Israel Institute of Technology.

Packetized GPS (PGPS) schemes), where each user *i* is associated with its fair share, x^i, where $x^i \neq x^j$ for users *i*, *j*. Most variants of the PGPS (or WFQ) schemes are computationally-intensive to implement, since they require keeping track of the finishing times (or times of departure) of packets in the corresponding (fluid) GPS scheme. However, under heavy load, where all users can be assumed to be continuously backlogged, the Weighted Round Robin (WRR) scheduler (which is *off-line*, and hence, independent of arrival characteristics) will be similar to the WFQ if packets are of fixed size [6]. Such a scheduler is simpler to implement and also analytically tractable.

In this study, we consider the design of off-line schedulers (such as WRR) and the analysis of their fairness performance for communications system, where each user comprises a stream of packets. Since the schedulers are off-line, the schedule (or assignment of time slots) is constructed according to a given $\underline{x}=\{x^j\}_{j=1}^K$, where x^j is the requested share of user *j*, $\sum_{j=1}^K x^j=1$ and $x^j \leq x^k$ if $j<k$. Such a schedule can be characterized in terms of its *fairness* and *smoothness*, which are defined as follows:

Fairness : A schedule is (*perfectly*) fair if the fraction of time slots allocated to each user *j* is (*equal*) close to x^j. This is similar to the notion of fairness over τ defined above.

Smoothness : A schedule is (*perfectly*) smooth if the time slots allocated to each user are (*equally spaced*) as evenly spaced as possible. This is similar to the notion of instantaneous fairness defined above.

A schedule that is *both* perfectly-fair and perfectly-smooth is desirable in many communication problems, for example,

QoS provisioning in TDMA-based wireless networks : In such networks, we envisage packets of different applications that differ in terms of data rates to be delivered from/to a wired network via a base station to/from wireless receivers within its coverage. The base station comprises multiple input queues, where queue *j* contains packets destined to wireless receiver *j* and its requested share, x^j, can be computed from the relative data rates of all queues. A scheduler is required at the base station to determine the queue to transmit at each time.

A schedule that is perfectly-fair ensures that each user receives its requested share (QoS). In addition, a smooth schedule minimizes the buffer requirement at the wireless receiver since the jitter of the inter-arrival intervals is minimized.

Data forwarding in wireless sensor networks : We consider wireless sensor networks where sensor nodes are grouped into clusters. The cluster-head of cluster *j* is responsible for aggregating the data collected within that cluster, and forwarding it to the base station, and its requested share, x^j, can be computed from the relative data arrival rates at all cluster heads. A scheduler is required at the base station to determine the cluster-head to transmit at each time.

A schedule that is perfectly-fair ensures that each cluster-head receives its requested share and this minimizes the probability of packet lost due to buffer overflow. In addition, a smooth schedule spreads out the transmissions, and hence, maximizes the lifetime of the nodes, which is the bottleneck in such networks.

However, a schedule that is *both* perfectly-fair and perfectly-smooth is infeasible for most $\underline{x}$. The design of perfectly-smooth schedulers can be found in [7, 8, 9, 10, 11] and the design of an *online* perfectly-fair scheduler that maximizes the throughput is considered in [12, 13, 14]. Given that the packet arrival process to each user is independent and identically distributed (i.i.d),

- for K=2 and unit buffer size per user [12], the schedule must be *open-loop* (or de-centralized) and *conflict-free*;
- for $K>2$, an optimal schedule always exists and is stationary and *cyclic* (or loop) [13], i.e., there exists an *R* such that for all *i*, the user allocated to slot *i* is also allocated to slot *i+R*;
- for $K>2$ and unlimited buffer size per user, the mean packet delay is minimized with a perfectly-smooth schedule [14]. A Golden Ratio Scheduler [14] is proposed that achieves a nearly-optimal throughput and performs extremely well compared to lower bounds for expected packet delay, although it is not perfectly-smooth.

As a result, the original online problem is reduced to an *offline* one, where the objective is to determine a loop schedule of size *R*, given $\underline{x}$, that is perfectly-fair and maximally smooth. However, no metric was proposed that measures the extent of non-smoothness of a given schedule.

In [15], the author defined the notion of *regularity* (equivalent to smoothness) for K=2. This notion is applied in [16], where the authors defined the equivalent notion of *balanced* schedules for $K>2$. Recently, the authors in [17] introduced a new notion of *m*-balancedness, where the value of *m* (which is a non-negative integer) gives a measure of smoothness of schedules for any *K*, where a smaller value indicates a more balanced schedule. The authors also proposed an *m*-balanced scheduler that constructs a schedule with $m \leq K$-1 for any given $\underline{x}$, but highlighted the difficulty in finding a schedule with the smallest possible *m*.

1.1 Perfectly-Fair Loop Schedulers

In this study, we focus on the special case where $\underline{x}$ comprises only rational elements. Then, we can write $\underline{x}$ as $[\frac{r^1}{R}, \frac{r^2}{R}, \cdots, \frac{r^K}{R}]$, where $\{r^j\}_{j=1}^K$ are positive integers, *R* = $\sum_{j=1}^K r^j$, and the greatest common divisor of $\{r^j\}_{j=1}^K$ is 1.

In this way, we can define $\mathbf{F}^{\underline{r}}$ as the class of *K*-flow perfectly-fair loop schedulers such that the following conditions are satisfied for any integer *z* and $1 \leq j \leq K$:

- r^j slots are allocated to user *j* over any interval of *R* slots (perfect-fairness over *R*);
- if slot *i* is allocated to user *j*, then slots $i+z \cdot R$ are also allocated to user *j*.

If $n_\pi^j(i)$ denotes the interval (in slots) between the $(i-1)^{th}$ and i^{th} allocation to user *j* under a scheduler π, then we have the following property:

Property 1 *If $\pi \in \mathbf{F}^{\underline{r}}$, then for $1 \leq j \leq K$,*

$$\begin{aligned} n_\pi^j(r^j + i) &= n_\pi^j(i) \\ \sum_{i=k}^{k+r^j-1} n_\pi^j(i) &= R, \text{ for any } k > 0 \end{aligned}$$

Hence, the elements, $\{n_\pi^j(i)\}_{i=1}^{r^j}$ (which we denote by $\underline{n}_\pi^j$) are sufficient to characterize any scheduler $\pi \in \mathbf{F}^{\underline{r}}$.

1.2 Class-based Scheduling Scenario

In any *K*-flow scheduling scenario given by $\underline{r}$, all users *j*,*k* with r^j=r^k can be grouped into the same class, based on the paradigm of service classes [18]. As a result, we propose an alternative class-based specification of the scenario given by $\underline{\kappa}$=$\{\kappa^c\}_{c=1}^C$, $\underline{\tilde{r}}$=$\{\tilde{r}^c\}_{c=1}^C$, such that class *c* comprises κ^c flows whose indices are defined by $\mathbf{C}^c$, where

$$\mathbf{C}^c = \{\sum_{y=1}^{c-1} \kappa^y + 1, \sum_{y=1}^{c-1} \kappa^y + 2, \cdots, \sum_{y=1}^{c} \kappa^y\}$$

$\tilde{r}^c$=r^j if $j \in \mathbf{C}^c$ and $\sum_{c=1}^C \kappa^c$=*K*. The special case of $\underline{\kappa}$=$\{1,\cdots,1\}$ corresponds to the original (class-less) scenario.

Such a class-based specification enables the definition of a *class-aware* scheduling paradigm, where slots are allocated to flows within each class *independently* of other classes (*intra*-class scheduling) and the allocation vectors obtained are subsequently *combined* in an optimal way (*inter*-class scheduling). This may result in a gain in performance and reduction in complexity over *class-unaware* schedulers.

1.3 Contribution of This Paper

In this paper, we consider the design of perfectly-fair loop schedulers (or loop schedulers in short). We propose a variance-based smoothness metric, and analyze the smoothness of the schedules obtained with various known loop schedulers. For a class-based scheduling scenario, we propose a recursive class-aware loop scheduler and demonstrate its performance gain over class-unaware loop schedulers.

The paper is organized as follows: We define our scheduling problem, where we consider the design of perfectly-fair loop schedulers to maximize allocation smoothness while maintaining intra-class fairness in a class-based scheduling scenario, in Section 2. We describe the mechanism and properties of several loop schedulers in Section 3. In Section 4, we propose a recursive loop scheduler that achieves good performance in terms of both smoothness and intra-class fairness in a class-based scheduling scenario. In Section 5, we compare the performance of various loop schedulers in terms of numerical results. Finally, some concluding remarks are given in Section 6.

2 Problem Definition

Given $\underline{r}$, our objective is to design a loop scheduler that generates a schedule that is as smooth as possible. In addition to *absolute* (ensemble) smoothness, *intra-class fairness* (which pertains to *relative* smoothness) is a desirable property in a class-based scheduling scenario. A loop scheduler is intra-class fair if the resulting schedule is equally smooth with respect to (i.e., has the same per-user smoothness for) all users in the same class, for all classes.

2.1 Smoothness Metrics

In this section, we describe metrics for evaluating per-user smoothness, from which the ensemble smoothness can be obtained. We propose a variance-based metric and then describe an existing metric that uses the notion of *m*-balancedness [17].

2.1.1 Variance-based metric

A schedule that is perfectly-smooth with respect to user *j* has equally-spaced allocations to user *j*, i.e., zero variance of $\underline{n}_\pi^j$ (denoted by $Var[\underline{n}_\pi^j]$). However, as the evenness (or smoothness) is reduced, $Var[\underline{n}_\pi^j]$ is increased. Hence, we define a metric (denoted by s_π^j) to evaluate the smoothness of a given schedule with respect to user *j* as follows:

$$s_\pi^j = Var[\underline{n}_\pi^j] = E[(\underline{n}_\pi^j)^2] - (E[\underline{n}_\pi^j])^2 \qquad (1)$$

where E[$(\underline{n}_\pi^j)^x$] = $\frac{\sum_{i=1}^{r^j} (n_\pi^j(i))^x}{r^j}$ is the x^{th} moment of $\underline{n}_\pi^j$ and according to Property 1, we have the following:

$$\begin{aligned} E[\underline{n}_\pi^j] &= \frac{\sum_{i=1}^{r^j} n_\pi^j(i)}{r^j} \\ &= \frac{R}{r^j} \text{ independent of } \pi \end{aligned}$$

Although s_π^j=0 with perfect-smoothness, the actual lower bound on s_π^j (denoted by s_{min}^j) for a given $\underline{r}$ is achieved according to the following theorem:

Theorem 1 *The minimum value of s_π^j for all $\pi \in \mathbf{F}^{\underline{r}}$ is achieved with $\underline{n}_{\pi^*}^j$ given as follows:*

$$\underline{n}_{\pi^*}^j = \{\overbrace{\lfloor\frac{R}{r^j}\rfloor,\cdots,\lfloor\frac{R}{r^j}\rfloor}^{r^j\lfloor\frac{R}{r^j}\rfloor+r^j-R},\underbrace{\lceil\frac{R}{r^j}\rceil,\cdots,\lceil\frac{R}{r^j}\rceil}_{R-r^j\lfloor\frac{R}{r^j}\rfloor}\} \quad (2)$$

where the order of the elements in $\underline{n}_{\pi^*}^j$ is unimportant, and its value is:

$$\begin{aligned} s_{\pi^*}^j &= s_{min}^j \\ &= \frac{R(2Q+1)-r^jQ(Q+1)}{r^j} - (\frac{R}{r^j})^2 \end{aligned}$$

where $Q = \lfloor\frac{R}{r^j}\rfloor$.

Proof. We consider the following cases:

R $\equiv$ **0 (modulo** r^j**) :** Perfect-smoothness is achieved for user *j* when the inter-allocation interval is constant, i.e., $n_{\pi^*}^j(k) = n_{\pi^*}^j(i)$. This is achieved if and only if $n_{\pi^*}^j(k) = \frac{R}{r^j}$ for $1 \leq k \leq r^j$.

R $\equiv$ ***y*** **(modulo** r^j**), 1**$\leq y \leq r^j$**-1 :** In this case, a constant inter-allocation interval for user *j* cannot be achieved. We thus have to find an optimal set of values for $n_{\pi^*}^j(k)$.

Let us define $\underline{n}_{\pi^*}^j$ as comprising *m* values, $\{a_k\}_{k=1}^m$, where each value a_k has multiplicity z_k, where a_k, $z_k \in \mathcal{Z}^+$. Without loss of generality, we can assume $a_v > a_w$ for *v*>*w*, which means that the set can be written as $\{a_1 + d_k\}_{k=1}^m$ and d_1=0, $d_v > d_w$ for *v*>*w*. Clearly, $\sum_{k=1}^m z_k$=r^j and $\sum_{k=1}^m z_k(a_1 + d_k)$=*R*.

The corresponding expression for E[$\underline{n}_{\pi^*}^j$]^2 is given as follows:

$$\begin{aligned} r^jE[(\underline{n}_{\pi^*}^j)^2] &= \sum_{k=1}^m z_k(a_1+d_k)^2 \\ &= \sum_{k=1}^m z_k d_k^2 + a_1(2R - a_1r^j) \\ &< \sum_{k=1}^m z_k d_k d_m + a_1(2R - a_1r^j) \\ &= d_m(R - a_1r^j) + a_1(2R - a_1r^j) \end{aligned}$$

It follows that to minimize E[$(\underline{n}_{\pi^*}^j)^2$], we have to minimize d_m, which is bounded from below by d_{m-1}. But setting d_m equal to d_{m-1} actually means reducing the dimensionality of $\underline{n}_{\pi^*}^j$ (it is interesting to note that the bound is independent of explicit z_k). Since d_1=0, we can continue this until *m*=2, where E[$(\underline{n}_{\pi^*}^j)^2$] is minimized by setting d_2=1. Hence, by setting *Q*=a_1, we have

$$\underline{n}_{\pi^*}^j = \{\overbrace{Q,\cdots,Q}^{z_1},\underbrace{Q+1,\cdots,Q+1}_{r^j-z_1}\} \quad (3)$$

Then, since $\sum_{k=1}^{r^j} n^j(k)$ = *R*, we have the following:

$$z_1 \cdot Q + (r^j - z_1)\cdot(Q+1) = R$$

from which we have

$$z_1 = r^j \cdot Q + r^j - R$$

However, since 1$\leq z_1 \leq$ r^j-1, we have the following constraints on *Q*:

$$\frac{R}{r^j} - 1 + \frac{1}{r^j} \leq Q \leq \frac{R}{r^j} - \frac{1}{r^j} \quad (4)$$

Since $\lfloor\frac{R}{r^j}\rfloor$ -1 < $\frac{R}{r^j} - 1 + \frac{1}{r^j}$ and $\lceil\frac{R}{r^j}\rceil > \frac{R}{r^j} - \frac{1}{r^j}$, the only integer *Q* that can satisfy Eq. (4) is *Q* = $\lfloor\frac{R}{r^j}\rfloor$.

Substituting these values into Eq. (3) results in the expression given in Eq. (2). The corresponding expression for s_{min}^j can be obtained by substituting $\underline{n}_{\pi^*}^j$ into Eq. (1). □

We define a general ensemble smoothness metric, s_π, in terms of $\{s_\pi^j\}_{j=1}^K$ and per-user weighting factors, $\{w^j\}_{j=1}^K$, where $\sum_{j=1}^K w^j$=1, as follows:

$$s_\pi = \sum_{j=1}^K w^j \cdot s_\pi^j \quad (5)$$

2.1.2 *m*-balancedness

Let $\underline{a}_\pi$ denote the schedule (comprising *R* slots) according to scheduler $\pi \in \mathbf{F}^{\underline{r}}$. If $\underline{y}$ denotes a sub-schedule (sequence of consecutive slots) in $\underline{a}_\pi$, then $|\underline{y}|$ is the length of $\underline{y}$, $j\underline{y}j$ is another sub-schedule that begins and terminates with *j* and $|\underline{y}|_j$ is the number of occurrences of *j* in $\underline{y}$.

According to [17], we have the following definition:

Definition 1 *For a non-negative integer m_π^j, a schedule, $\underline{a}_\pi$, is m_π^j-balanced with respect to user j if the following condition holds: for any sub-schedule, $j\underline{y}j$ in $\underline{a}_\pi$, any other sub-schedule, $\underline{y}'$ in $\underline{a}_\pi$ such that $|\underline{y}'| = |\underline{y}|+m_\pi^j+1$ satisfies $|\underline{y}'|_j \geq |\underline{y}|_j+1$.*

We note that m_π^j = 0 if $\underline{a}_\pi$ is perfectly-smooth with respect to user *j*; otherwise, the pseudocode for computing m_π^j is given below:

Algorithm for Computation of m_π^j, $1 \le j \le K$

Initialize $\{m_\pi^j\}_{j=1}^K = 0$
Set $a = b = 1$
while $a \le K$

$$\overline{M}(a,b) = \max_{1 \le u \le r^a} \sum_{y=u+1}^{u+b} n_\pi^a(y)$$

$$\underline{M}(a,b) = \min_{1 \le u \le r^a} \sum_{y=u+1}^{u+b} n_\pi^a(y)$$

$m_\pi^a = \max(\overline{M}(a,b)\text{-}\underline{M}(a,b), m_\pi^a)$
if $b = \lfloor \frac{r^a}{2} \rfloor$ or $\overline{M}(a,b)\text{-}\underline{M}(a,b) = 0$
    $a = a + 1$
else
    $b = b + 1$

According to the above algorithm, for a given $\underline{n}_\pi^j$, the value of m_π^j depends on the *order* of the elements in $\underline{n}_\pi^j$, and we have the following corollary to Theorem 1:

Corollary 1 *If $\underline{n}_{\pi^*}^j$ comprises the elements given in Eq. (2), then $m_{\pi^*}^j$ satisfies the following properties, where $V = \min\{r^j \lfloor \frac{R}{r^j} \rfloor + r^j\text{-}R, R\text{-}r^j \lfloor \frac{R}{r^j} \rfloor\}$:*

$$\begin{array}{ll} m = V, & V \le 2; \\ \lceil \frac{r^j+1}{2} \rceil - V \le m_{\pi^*}^j \le V, & V > 2. \end{array}$$

The lower and upper bounds for $m_{\pi^*}^j$ are obtained when $\underline{n}_{\pi^*}^j$ are given respectively as follows (where the order of the elements are important):

$$\underline{n}_{\pi^*}^j = [\overbrace{\lfloor \frac{R}{r^j} \rfloor, \cdots, \lfloor \frac{R}{r^j} \rfloor}^{r^j \lfloor \frac{R}{r^j} \rfloor + r^j - R}, \underbrace{\lceil \frac{R}{r^j} \rceil, \cdots, \lceil \frac{R}{r^j} \rceil}_{R - r^j \lfloor \frac{R}{r^j} \rfloor}]$$

$$\underline{n}_{\pi^*}^j = [\lfloor \frac{R}{r^j} \rfloor, \lceil \frac{R}{r^j} \rceil, \lfloor \frac{R}{r^j} \rfloor, \lceil \frac{R}{r^j} \rceil, \cdots, \lfloor \frac{R}{r^j} \rfloor, \lceil \frac{R}{r^j} \rceil]$$

As in Eq. (5), the corresponding ensemble smoothness metric, m_π, can be defined as follows:

$$m_\pi = \sum_{j=1}^{K} w^j \cdot m_\pi^j$$

A specific ensemble smoothness metric is proposed in [17] as follows:

$$m_\pi = \max_{1 \le j \le K} m_\pi^j \quad (6)$$

which corresponds to the following weighting factors:

$$w^j = \begin{cases} 1, & j = \arg\max_{1 \le i \le K} m_\pi^i; \\ 0, & \text{otherwise.} \end{cases}$$

2.1.3 Comparison between m_π^j and s_π^j

According to Sections 2.1.2 and 2.1.1, both metrics, m_π^j and s_π^j, can be computed given $\underline{n}_\pi^j$. However, according to Theorem 1 and Corollary 1, for the set of elements, $\underline{n}_{\pi^*}^j$, given in Eq. 2, while the value of $s_{\pi^*}^j$ is *unique*, the corresponding value of $m_{\pi^*}^j$ can only be given in terms of a *range*. This is true for any valid $\underline{n}_\pi^j$, i.e., s_π^j is unique while m_π^j depends on the *order* of the individual elements in $\underline{n}_\pi^j$. This imposes much more stringent conditions on $\underline{n}_\pi^j$ (and therefore, it is harder to construct a schedule) to achieve optimality in terms of m_π^j.

In the subsequent analysis, we will quantify the per-user smoothness of various loop schedulers in terms of the variance-based metric; the corresponding *m*-balancedness metric will be computed for the purpose of comparison.

2.2 Intra-class Unfairness Metric

Let d_π^c denote the number of *distinct* values of per-user smoothness (s_π^j or m_π^j) for all users $j \in \mathbf{C}^c$ with scheduler π, where $1 \le d_\pi^c \le \kappa^c$.

Since π is intra-class fair if $\underline{a}_\pi$ achieves the same per-user smoothness for all users in the same class, for all classes, i.e., d_π^c=1, $1 \le c \le C$, a larger value of d_π^c indicates larger intra-class unfairness for class *c*. Hence, we define the following measure to quantify the level of unfairness over the ensemble of all classes with scheduler π:

$$u_\pi = \frac{1}{K} \sum_{c=1}^{C} d_\pi^c$$

According to the definition of d_π^c, the unfairness metric, u_π, is bounded as follows:

$$\frac{C}{K} \le u_\pi \le 1$$

We use the notations $u_\pi(s)$ and $u_\pi(m)$ to denote the unfairness metrics based on s_π^j and m_π^j respectively.

2.3 Problem Formulation

Since our objective is to determine a scheduler that minimizes both the smoothness and intra-class unfairness metrics, it can be formulated as an optimization problem as follows:

***K*-flow Loop Scheduling Problem**

Determine the schedule $\underline{a}_{\pi^*}$ such that

$$s_{\pi^*} = \min_{\pi \in \mathbf{F}^{\underline{r}}} s_\pi$$

and

$$u_{\pi^*} = \min_{\pi \in \mathbf{F}^{\underline{r}}} u_\pi$$

To assess the size of the problem, let $\mathbf{A} = \{\underline{a}_\pi : \pi \in \mathbf{F}^{\underline{r}}\}$. Then, we have the following:

$$|\mathbf{A}| = \frac{R!}{\prod_{j=1}^{K} r^j!}$$

We note that a number of $\underline{a}_\pi \in \mathbf{A}$ are equivalent since they are identical under rotation with respect to the performance metrics. However, even after eliminating these, the resultant space is still non-tractable for large *R*.

A dynamic programming approach to derive an optimal scheduler requires the definition of an additive objective function, i.e., one which is computed incrementally. However, the per-flow smoothness metric, s_π^j, is a cumulative quantity, which renders the approach unsuitable.

Therefore, our approach is to evaluate the performance of various known loop schedulers in terms of both smoothness and intra-class unfairness, benchmarked against the respective lower bounds.

3 Description of *K*-flow Loop Schedulers

In this section, we describe the mechanism and study the per-user smoothness (according to the variance-based metric) and intra-class fairness properties of several loop schedulers.

3.1 *K*-flow Deficit Round Robin Scheduler (DRR_K)

Fair-queueing schedulers (e.g., Weighted-Fair Queueing (WFQ)) achieve nearly-perfect fairness, but they are usually expensive to implement. DRR_K [19] is an online scheduler that is an approximation to fair-queueing which is simple to implement and yet achieves good fairness and can also be implemented as a loop scheduler. Within the scope of our scheduling problem, the DRR_K scheduler reduces to a Weighted Round Robin (WRR) policy, which simply allocates a block of r^1 slots to user 1 followed by a block of r^2 slots to user 2 and so on. Hence, each user *j* is allocated slots in blocks of size r^j, with an interval of *R*-r^j slots between successive blocks. Therefore, we have the following:

$$\underline{n}^j_{DRR_K} = \{\overbrace{1, \cdots, 1}^{r^j-1}, R - r^j + 1\} \quad (7)$$

The performance of the DRR_K is given by the following theorem:

Theorem 2 *The DRR_K scheduler ensures intra-class fairness, but exhibits the worst smoothness amongst $\pi \in \boldsymbol{F}^{\underline{r}}$, i.e., for $1 \leq j \leq K$, its value is given by:*

$$\begin{aligned} s^j_{DRR_K} &= \frac{r^j + (R - r^j)^2 + 2(R - r^j)}{r^j} - (\frac{R}{r^j})^2 \\ &= \max_{\pi \in \boldsymbol{F}^{\underline{r}}} s^j_\pi \end{aligned} \quad (8)$$

Proof. The expression for $s^j_{DRR_K}$ given in Eq. (8) can be computed by substituting Eq. (7) into Eq. (1).

Let us consider an arbitrary scheduler $\pi \in \mathbf{F}^{\underline{r}}$ with $\underline{n}^j_\pi$ given as follows:

$$\underline{n}^j_\pi = \{1 + z_1, \cdots, 1 + z_{r^j-1}, R - r^j + 1 - \sum_{y=1}^{r^j-1} z_y\}$$

where $z_y \in \mathbf{Z}^+$, $1 \leq y \leq r^j$-1. We note that for z_y=0, $1 \leq y \leq r^j$-1, $\pi = DRR_K$.

We can express s^j_π in terms of $s^j_{DRR_K}$ as follows:

$$\begin{aligned} s^j_\pi &= s^j_{DRR_K} \\ &+ \frac{\sum_{y=1}^{r^j-1} z_y^2 + [\sum_{y=1}^{r^j-1} z_y]^2 - 2(R - r^j) \sum_{y=1}^{r^j-1} z_y}{r^j} \end{aligned} \quad (9)$$

Since $\underline{n}^j_\pi$ corresponds to inter-allocation intervals, we have the following constraint:

$$R - r^j + 1 - \sum_{y=1}^{r^j-1} z_y \geq 1$$

In addition, according to the triangular inequality, we have:

$$\sum_{y=1}^{r^j-1} z_y^2 \leq [\sum_{y=1}^{r^j-1} z_y]^2$$

Substituting into Eq. (9), we have the following:

$$\begin{aligned} s^j_\pi &\leq s^j_{DRR_K} + \frac{\sum_{y=1}^{r^j-1} z_y^2 + [\sum_{y=1}^{r^j-1} z_y]^2 - 2[\sum_{y=1}^{r^j-1} z_y]^2}{r^j} \\ &= s^j_{DRR_K} + \frac{\sum_{y=1}^{r^j-1} z_y^2 - [\sum_{y=1}^{r^j-1} z_y]^2}{r^j} \\ &\leq s^j_{DRR_K} \end{aligned}$$

According to Eq. (8), the per-user smoothness metric for user *j* is a function of r^j only and hence, the DRR_K scheduler ensures intra-class fairness. □

3.2 *K*-flow Credit Round Robin Scheduler (CRR_K)

The motivation to design the CRR_K scheduler [20] was to reduce the latency of the DRR_K scheduler. As with the DRR_K scheduler, the CRR_K scheduler can be implemented as a loop scheduler, and the pseudo-code is given as follows:

***K*-flow Credit Round Robin Scheduler (CRR_K)**

```
Initialize y^j = r^j/r^K, 1≤ j ≤ K
Set i=1, SP=K, count=0
while i ≤ R
    if count < K
      if y^SP < 1
        count = count + 1
      else
        a_CRR_K(i) = SP,
        y^SP = y^SP - 1, i = i + 1, count = 0
      SP = SP - 1 (modulo K)
    else y^j = y^j + r^j/r^K ∀ j, count = 0
```

The CRR_K scheduler possesses the following property for $1 \leq j \leq K$-1 [21]:

Property 2 *The i^{th} allocation of user j always occurs between the $\lceil \frac{ir^K}{r^j} \rceil$ th and $\lceil \frac{ir^K}{r^j} \rceil$ - 1 th allocation of user K, 1≤ i≤ r^j.*

Property 2 can be generalized for a class-based scenario as follows:

Property 3 *For the CRR_K scheduler, users within each class are allocated in blocks, where the order within class $\mathbf{C}^u$ is $\sum_{i=1}^{u-1} \kappa^i + 1$, $\sum_{i=1}^{u-1} \kappa^i$+2, $\cdots$, $\sum_{i=1}^{u} \kappa^i$ for 1≤ $u \leq C$. In addition, the i^{th} block of $\mathbf{C}^u$ will reside between the $\lceil \frac{i \cdot \tilde{r}^C}{\tilde{r}^u} \rceil^{th}$ and $\lceil \frac{i \cdot \tilde{r}^C}{\tilde{r}^u} \rceil - 1^{th}$ block of $\mathbf{C}^C$, where 1≤ $i \leq \tilde{r}^u$.*

We note from Property 3 that users within each class are always transmitted in blocks, where each user from that class is allocated exactly once and the order within each block is constant. Hence, the per-user smoothness for users belonging to the same class are identical, i.e., intra-class fairness is maintained.

3.3 *K*-flow Smoothed Round Robin Scheduler (SRR_K)

The SRR_K scheduler [22] is a variant of the standard WRR scheduler, aimed at reducing the latter's output burstiness and short-term unfairness (i.e., improving allocation smoothness). A Weight Spread Sequence ($\underline{S}$) that distributes the allocation to each user evenly and a Weight Matrix ($\underline{\underline{M}}$) that is a binary representation of $\underline{r}$ are two key structures of the scheduler. The pseudo-code for the SRR_K scheduler is described as follows, where the function $dec2bin(i,k)$ converts the integer *i* into its binary representation with *k* bits:

***K*-flow SRR Scheduler (SRR_K)**

r_{max} = $\max_{1 \leq j \leq K} r^j$
k = $\lceil log_2(r_{max}+1) \rceil$
$\underline{S}$ = []
for *i*=1:*k*
  $\underline{S}$ = [$\underline{S}$ *i* $\underline{S}$]
$\underline{\underline{M}}$ = []
for *j*=1:*K*
  $\underline{\underline{M}}$ = [$\underline{\underline{M}}$; *dec2bin*($r^j$,*k*)]
$\underline{a}$=[]
for *i*=1:length($\underline{S}$)
  index = find($\underline{\underline{M}}$(:,S(i))==1)
  $\underline{a}$ = [$\underline{a}$ index]

According to the above algorithm, the SRR_K scheduler possesses the following property for a class-based scenario:

Property 4 *For the SRR_K scheduler, users within each class are allocated in blocks, where the order within class $\mathbf{C}^u$ is $\sum_{i=1}^{u-1} \kappa^i + 1$, $\sum_{i=1}^{u-1} \kappa^i$+2, $\cdots$, $\sum_{i=1}^{u} \kappa^i$ for 1≤ $u \leq C$.*

As with the CRR_K scheduler, we can deduce from Property 4 that intra-class fairness is maintained for the SRR_K scheduler.

3.4 *K*-flow Weighted Round Robin with WFQ-like spreading Scheduler (*WRR*-sp_K)

The *WRR*-sp_K scheduler [23] is another variant of the standard WRR scheduler, in which the service order amongst the users is identical to WFQ. The algorithm for the *WRR*-sp_K scheduler is described as follows:

***K*-flow WRR with spreading Scheduler (*WRR*-sp_K)**

Let the array $\underline{y}$ contain the sequence
$< \frac{q}{r^j}, j >$: $q \in \{1, \cdots, r^j\}$, 1≤ $j \leq K$
sorted in lexicographic order.
The vector $\underline{a}_{WRR-sp_K}$ is constructed as follows:
  $a_{WRR-sp_K}(i)$ = *j* if $y(i) = < \frac{q}{r^j}, j >$

According to the algorithm, the sequence $< \frac{q}{r^j}, j >$ is sorted in *lexicographic* order, i.e., they are sorted in ascending order (*primary* sorting) according to the first component (i.e., $\frac{q}{r^j}$) of each element ($\frac{q}{r^j}$,*j*) and in the event of a tie, the elements will be sorted in ascending order (*secondary* sorting) according to the second component (i.e., *j*).

The *WRR*-sp_K scheduler possesses the following property for 1≤ $j \leq K$-1 [21]:

Property 5 *The i^{th} allocation of user j always occurs between the $\lceil \frac{ir^k}{r^j} \rceil$ th and $\lceil \frac{ir^k}{r^j} \rceil$ - 1 th allocation of user k, where k>j, 1≤ i≤ r^j.*

Property 5 can be generalized for a class-based scenario as follows:

Property 6 *For the WRR-sp_K scheduler, users within each class are allocated in blocks, where the order within class $\mathbf{C}^u$ is $\sum_{i=1}^{u-1} \kappa^i + 1$, $\sum_{i=1}^{u-1} \kappa^i$+2, $\cdots$, $\sum_{i=1}^{u} \kappa^i$ for 1≤ $u \leq C$. In addition, the i^{th} block of $\mathbf{C}^u$ will reside between the $\lceil \frac{i \cdot \tilde{r}^y}{\tilde{r}^u} \rceil^{th}$ and $\lceil \frac{i \cdot \tilde{r}^y}{\tilde{r}^u} \rceil - 1^{th}$ block of $\mathbf{C}^y$, where y>u and 1≤ $i \leq \tilde{r}^u$.*

As with the CRR_K scheduler, we can deduce from Property 6 that intra-class fairness is maintained for the *WRR*-sp_K scheduler.

We propose a generic *WRR*-sp_K scheduler (denoted by *WRR*-$sp_K(\varrho)$, 1≤ $\varrho \leq K$) such that the secondary sorting is performed in the order [ϱ,ϱ+1,ϱ+2,$\cdots$,*K*,1,2,$\cdots$,ϱ-1]. We define a maximally-smooth *WRR*-$sp_K(\varrho)$ scheduler (denoted by $WRR-sp_K^*$) as follows:

$$s_{WRR-sp_K^*} = \min_{1 \leq \varrho \leq K} s_{WRR-sp_K(\varrho)}$$

We note that Property 6 is preserved with the $WRR-sp_K^*$ scheduler.

3.5 *K*-flow Golden Ratio (GR_K) Scheduler

The Golden Ratio Scheduler was proposed in [13] and is described as follows:

***K*-flow Golden Ratio Scheduler (GR_K)**

Let z = 0.6180339887 and $w(m) = frac(m \cdot z)$
where $frac(y) = y - \lfloor y \rfloor$
Let the array $\underline{y}$ contain the sequence $w(m)$,
$0 \leq m \leq R$-1, sorted in ascending order.
The vector $\underline{a}_{GR_K}$ is constructed as follows:
$a_{GR_K}(i)$ = *j*
if $\sum_{k=1}^{j-1} \frac{r^k}{R} \leq y(i) \leq \sum_{k=1}^{j} \frac{r^k}{R}$, $1 \leq j \leq K$

It was established in [14] that if *R* is a Fibonacci number, then $\underline{n}^j_{GR_K}$ comprises at most three values for each *j*; otherwise, more values are generated.

3.6 *K*-flow Short-term Fair Scheduler (STF_K)

We can characterize the throughput-*fairness* (as opposed to the notion of intra-class fairness defined in Section 2 that pertains to smoothness) of any loop scheduler in terms of the *cumulative service-deficit*, $sd^j(i)$, which measures the discrepancy between the requested and allocated share for user *j* up to slot *i*, $1 \leq i \leq R$. If $y^j(i)$ denotes the cumulative number of slots allocated to user *j* up to and including slot *i*, then we have the following:

$$sd^j(i) \quad = \quad \frac{r^j}{R} - \frac{y^j(i)}{i}$$

A *positive* (negative) value of $sd^j(i)$ implies that user *j* has received *less* (more) than its requested share up to slot *i*. Hence, we consider a scheduler that allocates each slot to the user with maximum instantaneous service-deficit so as to achieve maximum throughput-fairness (Short-term Fair or STF_K scheduler). Whenever there is a tie, priority for allocation is given to the user with the highest flow index. The pseudo-code for the STF_K scheduler is given as follows:

***K*-flow Short-term Fair Scheduler (STF_K)**

Initialize $y^j(0) = 0$, $1 \leq j \leq K$
for *i*=1:*R*
$y^j(i) = y^j(i-1)$, $1 \leq j \leq K$
$sd^j(i) = \frac{r^j}{R} - \frac{y^j(i)}{i}$, $1 \leq j \leq K$
$a_{STF_K}(i) = \arg \max_{1 \leq j \leq K} sd^j(i)$
$y^{a_{STF_K}(i)}(i) = y^{a_{STF_K}(i)}(i)+1$

This scheduler was first suggested in [13], where the authors conjectured, based on numerical calculations, that it is a promising scheduler. However, no analysis of the scheduler was provided in terms of smoothness.

According to our analysis [21], the STF_K scheduler possesses the following property:

Property 7 *For any two-class scheduling scenario (C=2), users within class* $\boldsymbol{C}^2$ *are allocated in blocks, where the order within each block is K,K-1,K-2,*$\cdots$*,K-*κ^1*; Users within class* $\boldsymbol{C}^1$ *are always allocated in the order* κ^1*,*κ^1*-1,*$\cdots$*,1 and the maximum number of users in class* $\boldsymbol{C}^1$ *allocated between two successive* $\boldsymbol{C}^2$ *blocks is* κ^1*.*

We can deduce from Property 7 that intra-class fairness is maintained for two-class scheduling with the STF_K scheduler.

3.7 *K*-flow *m*-Balanced Scheduler ($MBAL_K$)

The *m*-balanced scheduler was proposed in [17] and its pseudocode is given as follows:

***K*-flow *m*-Balanced Scheduler ($MBAL_K$)**

Initialize $\phi = \{\phi^1, \phi^2, \cdots, \phi^K\}$ s.t.
ϕ^j is uniformly distributed on [0, $\frac{R}{r^j}$), $1 \leq j \leq K$
for *i* = 1:*R*
y = $\arg \min_{1 \leq j \leq K} \phi$
$a_{MBAL_K}(i)$ = *y*
$\phi^y = \phi^y + \frac{R}{r^y}$

The smoothness metric corresponding to the schedule constructed by the algorithm is upper-bounded according to the following property:

Property 8 *According to the metric defined in Eq. (6), the worst-case m-balancedness of the* $MBAL_K$ *scheduler is K-1, i.e.,* $m_{MBAL_K} \leq$ *K-1.*

It is difficult to find an initial value for ϕ that achieves the best possible smoothness. Hence, the authors proposed iterating the algorithm over a predetermined number of runs (denoted by *ITER*) in an attempt to improve the schedule.

3.8 *K*-flow Random (RND_K) Scheduler

The loop schedulers considered so far are *deterministic* since the schedule $\underline{a}_\pi \in \mathbf{A}$ is fixed. In this section, we define a *random* scheduler, RND_K, whose schedule, $\underline{a}_{RND_K}$, is *uniformly* selected from $\mathbf{A}$. We note that $RND_K \in \mathbf{F}^{\underline{r}}$ because the selected $\underline{a}_{RND_K}$ is used for allocation in each loop.

The per-user smoothness metric for each user *j* is given as follows [21]:

$$s^j_{RND_K} \quad = \quad \frac{R(2R - r^j + 1)}{r^j(r^j + 1)} - (\frac{R}{r^j})^2 \qquad (10)$$

By comparing Eq. (10) with Theorem 2, we have the following corollary:

Corollary 2 *The per-user smoothness metric for the* RND_K *scheduler is upper-bounded by that of the* DRR_K *scheduler, i.e., for* $1 \leq j \leq K$*,*

$$s^j_{RND_K} \quad \leq \quad s^j_{DRR_K}$$

4 Design of Class-aware Loop Scheduler

Our analysis in Section 3 suggests that the GR_K, $MBAL_K$ and RND_K schedulers do not ensure intra-class fairness, which is a desirable property in a class-based scheduling scenario. On the other hand, while the STF_K scheduler guarantees intra-class fairness only for a two-class scenario (*C*=2), the CRR_K, SRR_K and $WRR - sp_K$ schedulers are intra-class fair for any *C*. Hence, the latter schedulers are suitable as bases for constructing *class-aware* loop schedulers. We begin with the design for *C*=2, and then extrapolate the design to the multi-class (*C*>2) scenario.

4.1 An Optimal Two-Class Loop Scheduler (*C*=2)

Comparing Property 3 and Property 6, it can be deduced that the CRR_K and $WRR - sp_K$ are equivalent for two-class scheduling, and hence, we restrict our consideration to the latter scheduler.

Using Property 6 and Theorem 1, $\underline{n}^j_{WRR-sp_K}$ and $\underline{n}^j_{\pi^*}$ can be computed and are given in Eq. (11) and (12) respectively on the next page. Comparing these equations, we note that $\underline{n}^j_{WRR-sp_K} \neq \underline{n}^j_{\pi^*}$ for $1 \leq j \leq K$ and hence, the *WRR*-sp_K scheduler is not optimal in terms of per-user smoothness. However, when κ^1=1 (κ^1=*K*-1), the *WRR*-sp_K scheduler offers optimal per-user smoothness for users in $\mathbf{C}^2$ ($\mathbf{C}^1$). Hence, since κ^1=1 =*K*-1 when *K*=2, the *WRR*-sp_2 scheduler offers optimal smoothness for *all* users, i.e.,

$$s^j_{WRR-sp_2} = \min_{\pi \in \mathbf{F}^{\underline{r}}} s^j_\pi$$

Hence, we can construct a class-aware scheduler (see Section 1.2) for a two-class scenario by (a) defining an equivalent two-flow scenario (K=2) and (b) constructing a schedule using the *WRR*-sp_2 scheduler (inter-class scheduling) and (c) substituting for the indices of the users (intra-class scheduling). Denoted by OPT_2, it can be shown that $\underline{n}^j_{OPT_2} = \underline{n}^j_{\pi^*}$ as given in Eq. (12) for $1 \leq j \leq K$, i.e., the OPT_2 scheduler achieves optimal smoothness for two-class scheduling. The pseudo-code is given below, assuming $\kappa^1 \cdot \tilde{r}^1 \leq \kappa^2 \cdot \tilde{r}^2$ (the corresponding scheduler for $\kappa^1 \cdot \tilde{r}^1 > \kappa^2 \cdot \tilde{r}^2$ can be obtained by interchanging the indices 1 and 2):

Optimal Two-Class Loop Scheduler (OPT_2)

Set $\underline{r} = [\kappa^1 \cdot \tilde{r}^1,\ \kappa^2 \cdot \tilde{r}^2]$

Define $\mathbf{I}^1 = [\overbrace{\mathbf{C}^1, \cdots, \mathbf{C}^1}^{\tilde{r}^1}]$, $\mathbf{I}^2 = [\overbrace{\mathbf{C}^2, \cdots, \mathbf{C}^2}^{\tilde{r}^2}]$

Compute $\underline{a}_{WRR-sp_2}$ = *WRR*-sp_2($\underline{r}$)

for *y* = 1:2

  index = find($\underline{a}_{WRR-sp_2}$==*y*)

  a_{OPT_2}(index) = $\mathbf{I}^y$

4.2 A Recursive Class-aware Loop Scheduler for Multi-class Scenario (*C*>2)

In this section, we construct a class-aware scheduler for a multiple-class scenario (*C*>2). In order to exploit the smoothness property of the optimal two-class scheduler (OPT_2), we propose a *recursive* approach that (a) partitions the original *C*-class problem into smaller sub-problems at various levels (*Forward*) (b) solves each sub-problem, beginning with the lowest level (*Solution*) and (c) substitutes the solutions obtained in the return path to obtain the required schedule for the original problem (*Return*).

We describe the approaches for each stage as follows, where the notation *REC*($\underline{I}$), $\underline{I} \in \underline{C}$={1,2,$\cdots$,*C*}, represents an $|\underline{I}|$-class scheduling problem.

4.2.1 Forward

We begin by partitioning the *C*-class (level 0) problem, $REC(\underline{C})$, into level 1 sub-problems, denoted by $REC(\underline{I}^1_1), REC(\underline{I}^1_2)$, where $\underline{I}^1_1 \cup \underline{I}^1_2 = \underline{C}$ and $\underline{I}^1_1 \cap \underline{I}^1_2 = \{\}$.

As far as this stage is concerned, we can interpret $REC(\underline{I}^1_1), REC(\underline{I}^1_2)$ as comprising two independent problems, $REC(\underline{I}^1_1)$ and $REC(\underline{I}^1_2)$, each of which can be further partitioned into level 2 sub-problems. For example, $REC(\underline{I}^1_1)$ can be partitioned into $REC(\underline{I}^2_1), REC(\underline{I}^2_2)$, where $\underline{I}^2_1 \cup \underline{I}^2_2 = \underline{I}^1_1$ and $\underline{I}^2_1 \cap \underline{I}^2_2 = \{\}$.

One particular approach for partitioning [21] is that for each level *y* sub-problem, $REC(\underline{I}^y_1), REC(\underline{I}^y_2)$, $\forall\ y \geq 1$, $\min\{|\underline{I}^y_1|, |\underline{I}^y_2|\}$=1.

4.2.2 Solution

Let us consider the level *y* sub-problem, $REC(\underline{I}^y_1), REC(\underline{I}^y_2)$. In order to solve this problem, we first compute $a_{REC(\underline{I}^y_1)}$ and $a_{REC(\underline{I}^y_2)}$ independently (intra-class scheduling), and then combine these schedules to obtain $a_{REC(\underline{I}^y_1),REC(\underline{I}^y_2)}$ (inter-class scheduling). The respective functions are described as follows:

intra-class scheduling : According to Section 4.1, the OPT_2 scheduler achieves optimal smoothness for any two-class scheduling scenario. Hence, the *forward* phase of the recursive scheduler is executed until level *l*, such that for each level *l* sub-problem, $REC(\underline{I}^l_1), REC(\underline{I}^l_2)$, $\max\{|\underline{I}^l_1|, |\underline{I}^l_2|\}$ = 2.

As such, we begin the *solution* phase at level *l*, and the resulting schedules, $a_{REC(\underline{I}^l_1)}$ and $a_{REC(\underline{I}^l_2)}$, will be optimally smooth. We note that a simple Round-Robin scheduler suffices to ensure optimal smoothness for a one-class scenario, i.e., $|\underline{I}^l|$=1.

inter-class scheduling : There are many different ways to combine $a_{REC(\underline{I}^y_1)}$ and $a_{REC(\underline{I}^y_2)}$ to obtain $a_{REC(\underline{I}^y_1),REC(\underline{I}^y_2)}$. One approach is to construct an equivalent two-user scheduling problem by aggregating the users in $\underline{I}^y_1$ and $\underline{I}^y_2$ respectively, and apply the *WRR*-sp_2 scheduler to obtain a two-user

$$\underline{n}^j_{WRR-sp_K} = \begin{cases} \overbrace{\{\kappa^1+\kappa^2\lfloor\frac{\tilde{r}^2}{\tilde{r}^1}\rfloor,\cdots,\kappa^1+\kappa^2\lfloor\frac{\tilde{r}^2}{\tilde{r}^1}\rfloor}^{\tilde{r}^1\lceil\frac{\tilde{r}^2}{\tilde{r}^1}\rceil-\tilde{r}^2}, \underbrace{\kappa^1+\kappa^2\lceil\frac{\tilde{r}^2}{\tilde{r}^1}\rceil\cdots\kappa^1+\kappa^2\lceil\frac{\tilde{r}^2}{\tilde{r}^1}\rceil}_{\tilde{r}^2-(\lceil\frac{\tilde{r}^2}{\tilde{r}^1}\rceil-1)\tilde{r}^1}\}, & j\in\mathbf{C}^1; \\ \overbrace{\{\kappa^2,\cdots,\kappa^2}^{\frac{\tilde{r}^2K-R}{\kappa^1}}, \underbrace{K,\cdots,K}_{\frac{R-\kappa^2\tilde{r}^2}{\kappa^1}}\}, & j\in\mathbf{C}^2. \end{cases} \quad (11)$$

$$\underline{n}^j_{\pi^*} = \begin{cases} \overbrace{\{\kappa^1+\lfloor\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rfloor,\cdots,\kappa^1+\lfloor\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rfloor}^{\tilde{r}^1(\kappa^1+\lfloor\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rfloor)+\tilde{r}^1-R}, \underbrace{\kappa^1+\lceil\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rceil,\cdots,\kappa^1+\lceil\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rceil}_{R-\tilde{r}^1(\kappa^1+\lfloor\frac{\kappa^2\tilde{r}^2}{\tilde{r}^1}\rfloor)}\}, & j\in\mathbf{C}^1; \\ \overbrace{\{\kappa^2+\lfloor\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rfloor,\cdots,\kappa^2+\lfloor\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rfloor}^{\tilde{r}^2(\kappa^2+\lfloor\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rfloor)+\tilde{r}^2-R}, \underbrace{\kappa^2+\lceil\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rceil,\cdots,\kappa^2+\lceil\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rceil}_{R-\tilde{r}^2(\kappa^2+\lfloor\frac{\kappa^1\tilde{r}^1}{\tilde{r}^2}\rfloor)}\}, & j\in\mathbf{C}^2. \end{cases} \quad (12)$$

schedule, $\underline{a}$. The slots allocated to user '1' and '2' are substituted with the indices of $\underline{I}^y_1$ and $\underline{I}^y_2$ respectively. The pseudo-code (similar to the OPT_2 scheduler) is given below:

Function $\underline{a}$ = ***inter-c***($\underline{a}_{\underline{I}^y_1}$,$\underline{a}_{\underline{I}^y_2}$)

Set $\underline{r}$ = [$\sum_{c\in\underline{I}^y_1}\kappa^c\cdot\tilde{r}^c$, $\sum_{c\in\underline{I}^y_2}\kappa^c\cdot\tilde{r}^c$]
Compute $\underline{a}_{WRR-sp_2}$ = *WRR-sp*$_2$($\underline{r}$)
for *q* = 1:2
  index = find($\underline{a}_{WRR-sp_2}$==*q*)
  a(index) = $\underline{a}_{\underline{I}^y_q}$

An alternative approach for inter-class scheduling (denoted by *inter-c*') is proposed in [21], and it is applicable if the following condition holds:

$$\min\{|\underline{I}^y_1|,|\underline{I}^y_2|\} = 1 \quad (13)$$

To illustrate the approach, let us assume that $|\underline{I}^y_2|$=1, and $|\underline{a}_{REC(\underline{I}^y_2)}| \le |\underline{a}_{REC(\underline{I}^y_1)}|$. If we let $\underline{v}$ = $\underline{a}_{REC(\underline{I}^y_2)}$ and $\underline{z} = \underline{a}_{REC(\underline{I}^y_1)}$, then our objective is to insert the elements of $\underline{z}$ into $\underline{v}$ so that successive elements of $\underline{v}$ are as uniformly-spaced as possible in the combined vector, $a_{REC(\underline{I}^y_1),REC(\underline{I}^y_2)}$, i.e., we attempt to maximize smoothness with respect to $\underline{v}$. This is illustrated in Fig. 1, where P=$\lceil\frac{\underline{z}}{\underline{v}}\rceil$ and Q is given as follows:

$$Q = \begin{cases} P, & |\underline{z}| = |\underline{v}|; \\ P-1, & \text{otherwise.} \end{cases}$$

For the case where $|\underline{v}| > |\underline{z}|$, we simply swap the definitions of $\underline{v}$ and $\underline{z}$.

4.2.3 Return

Upon completion of the solution phase with all level *l* sub-problems, we begin the *return* phase, which computes the schedule for each level *y*-1 sub-problem using the solutions of its level *y* sub-problems iteratively until we arrive at the schedule for the original problem.

Let us consider a level *l*-1 sub-problem, $REC(\underline{I}^{l-1}_1), REC(\underline{I}^{l-1}_2)$. Assume that $REC(\underline{I}^{l-1}_1)$ is partitioned into level *l* sub-problems, $REC(\underline{I}^l_1), REC(\underline{I}^l_2)$ and $REC(\underline{I}^l_3), REC(\underline{I}^l_4)$. We apply the solution phase on each of the above level *l* sub-problems to obtain the respective schedules:

$$S^{l-1}_1 = \{a_{REC(\underline{I}^l_1),REC(\underline{I}^l_2)}; a_{REC(\underline{I}^l_3),REC(\underline{I}^l_4)}\}$$

Similarly, $REC(\underline{I}^{l-1}_2)$ is partitioned into level *l* sub-problems, $REC(\underline{I}^l_5), REC(\underline{I}^l_6)$ and $REC(\underline{I}^l_7), REC(\underline{I}^l_8)$, and the respective solutions are given as follows:

$$S^{l-1}_2 = \{a_{REC(\underline{I}^l_5),REC(\underline{I}^l_6)}; a_{REC(\underline{I}^l_7),REC(\underline{I}^l_8)}\}$$

In order to obtain $\underline{a}_{REC(\underline{I}^{l-1}_1),REC(\underline{I}^{l-1}_2)}$, we have to determine the combination, $(\underline{a}_1 \in S^{l-1}_1, \underline{a}_2 \in S^{l-1}_2)$, that results in a schedule, *inter-c*($\underline{a}_1$,$\underline{a}_2$), with the best smoothness, i.e., if $s_{\underline{a}}$ is the smoothness metric corresponding to the schedule $\underline{a}$, then:

$$s_{inter-c(\underline{a}_1,\underline{a}_2)} = \arg\min_{\underline{b}_1\in S^{l-1}_1,\underline{b}_2\in S^{l-1}_2} s_{inter-c(\underline{b}_1,\underline{b}_2)} \quad (14)$$

Once we have obtained the required schedules for all level *l*-1 sub-problems, they are returned to the corresponding level *l*-2 sub-problems in the same way until we arrive at the original problem.

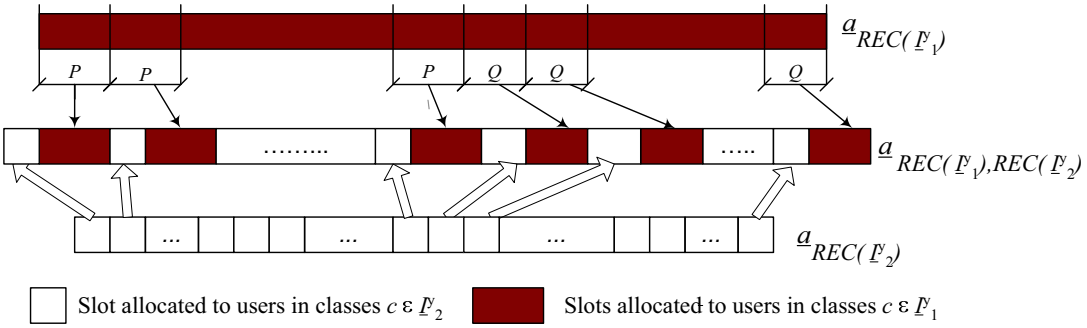

Figure 1: Illustration of the inter-class scheduler, *inter-c*'(), for the *REC*($\underline{C}$) scheduler.

4.3 Variants of Recursive Class-aware Loop Schedulers

According to Section 4.2, we may define variants of the $REC(\underline{C})$ that differ in terms of (a) the approach for partitioning in the *forward* phase and (b) the inter-class scheduling function in the *solution* phase. In this section, we define variants considered in this study for *C*=3 and *C*=4.

***C*=3 :** In this case, there is only one approach for partitioning, and this is illustrated in Fig. 2. However, since each sub-problem satisfies Eq. (13), we can define two variants (denoted by $REC_1(\underline{C})$ and $REC_2(\underline{C})$) that employs *inter-class*() and *inter-class*'() respectively for inter-class scheduling.

***C*=4 :** In this case, two approaches for partitioning exist, as illustrated in Fig. 3 (a) and (b) respectively. With approach (a), only *inter-class*() is valid, and we denote the resulting scheduler as $REC_1(\underline{C})$. On the other hand, with approach (b), since each sub-problem satisfies Eq. (13), we can define two variants (denoted by $REC_2)(\underline{C})$ and $REC_3(\underline{C})$) that employs *inter-class*() and *inter-class*'() respectively for inter-class scheduling.

We summarize the properties of variants of the $REC(\underline{C})$ scheduler for *C*=3 and *C*=4 in Table 1. In general, the computational requirement of the $REC(\underline{C})$ is upper-bounded by the variant where each sub-problem satisfies Eq. 13, and the algorithm requires $\binom{C}{2}$ runs of $OPT_2()$, $C(2^{C-1}\text{-}C)$ runs of *inter-c*() and $1 + \sum_{q=0}^{C-4} \prod_{i=C-q}^{C} i$ runs of *min*() (Eq. (14)). The fact that *C* is usually small makes the problem tractable in practical cases.

5 Numerical Results

We consider the following broadband applications with the corresponding typical bandwidth requirements in kbps [24]: Streaming Video (Internet Quality) (128), Residential Voice (300), Video Telephony (400) and Interactive Games (500).

We define various *C*-class scheduling scenarios, where $C \in \{3, 4\}$, and the user composition is assumed to be uniform, i.e., $\kappa^c = \kappa$ for $1 \leq c \leq C$. For *C*=4, each class comprises users from each of the above applications, and the scheduling scenario is defined by $\underline{\tilde{r}}$ = [128,300,400,500] = [32,75,100,125]. For *C*=3, multiple scheduling scenarios are possible. For example, if we consider Residual Voice, Video Telephony and Interactive Games, then we have $\underline{\tilde{r}}$=[300,400,500] $\equiv$ [3,4,5].

The weighting factors, $\{w^j\}_{j=1}^{K}$, are chosen to be proportional to the relative demand of each user as follows:

$$w^j = \frac{r^j}{R}$$

For the $MBAL_K$ scheduler, we set the number of iterations, *ITER*, to 1000. This number has to be sufficiently large so that it is more likely to obtain the best possible scheduler; however, this is achieved at the expense of increased computational complexity.

5.1 Performance of Variants of Recursive Class-aware Loop Schedulers

We begin by comparing the smoothness (s_π) and intra-class unfairness (u_π) achieved with the variants of recursive class-aware loop schedulers defined in Section 4.3. The results for the 4-class scheduling scenario, $\underline{\tilde{r}}$ = [32,75,100,125], are plotted in Fig. 4. The corresponding results for the 3-class scheduling scenario, $\underline{\tilde{r}}$=[32,75,125], are shown in Fig. 5.

***C*=4 :** In terms of smoothness, using the partitioning approach that satisfies Eq. 13 (i.e., *REC*-2 and *REC*-3), we note that *REC*-2 (that uses *inter-c*()) achieves better performance than *REC*-3 (that uses *inter-c*'()). This is expected since *inter-c*() is derived from the *OPT*-2 scheduler, which achieves optimal smoothness for two-class scheduling.

Comparing between the partitioning approaches, the *REC*-2 scheduler performs better than the *REC*-1 scheduler at the expense of computational complexity, since the *REC*-2 scheduler is computationally more expensive (See Table 1).

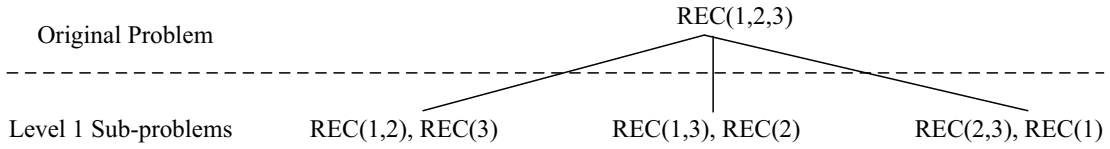

Figure 2: Partitioning approach for the $REC(\underline{C})$ scheduler (C=3).

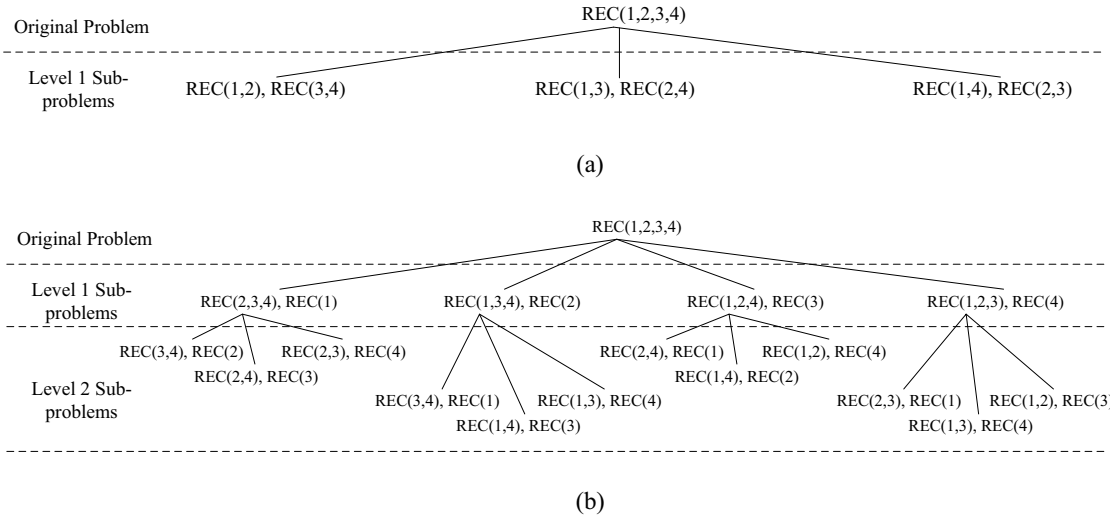

Figure 3: Approaches for partitioning for the $REC(\underline{C})$ scheduler (C=4).

C	Variants of REC(C)	Partitioning approach	Inter-class scheduler	Computational Requirements (No of runs)		
				$OPT_2()$	inter-c() or inter-c'()	min() in Eqn (6)
3	REC-1	All sub-problems satisfy Eqn (5)	inter-c()	3	3	1
	REC-2		inter-c'()			
4	REC-1	No sub-problem satisfies Eqn (5)	inter-c()	6	3	1
	REC-2	All sub-problems satisfy Eqn (5)	inter-c()	6	16	5
	REC-3		inter-c'()			

Table 1: Properties of variants of $REC(\underline{C})$ scheduler (C=3,4)

In terms of fairness, *REC*-1 and *REC*-2 are less unfair than *REC*-3, which once again demonstrates the superiority of *inter-c*() over *inter-c*'() (implemented in *REC*-3).

***C*=3 :** Since the partitioning approach is unique, the recursive schedulers are distinguished according to the inter-class scheduling function implemented. We note that *REC*-1 (that uses *inter-c*()) achieves better smoothness performance than *REC*-2 (that uses *inter-c*'()). This concurs with the observations for *C*=4.

In terms of fairness, *REC*-1 is less unfair than *REC*-2, which once again demonstrates the superiority of *inter-c*() over *inter-c*'() (implemented in *REC*-2).

Table 2 summarizes the comparison between both variants of recursive schedulers for 3-class scheduling scenarios. The above observations for *C*=3 are consistent over the various 3-class scheduling scenarios.

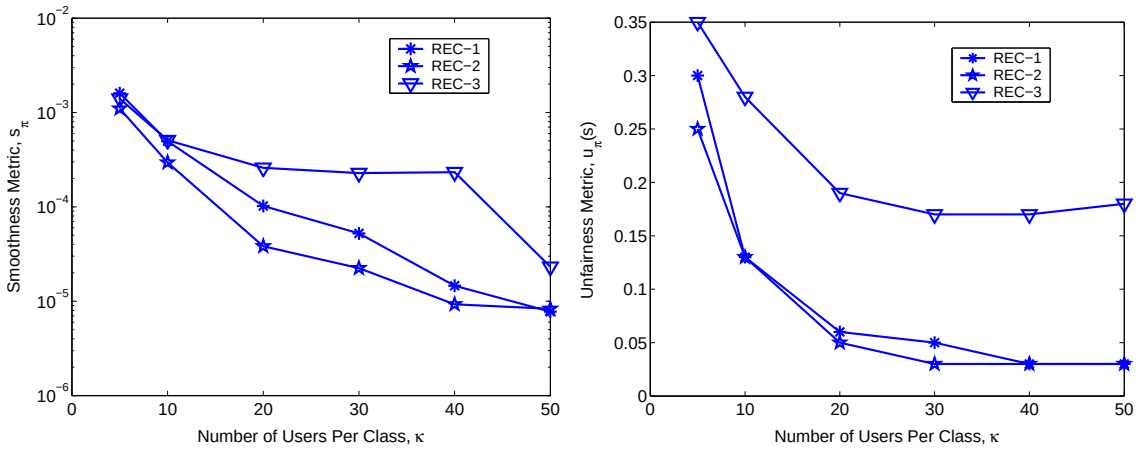

Figure 4: Comparison of s_π (left) and u_π (right) amongst variants of recursive loop schedulers for $\underline{\tilde{r}}$ = [32,75,100,125] in 4-class scheduling.

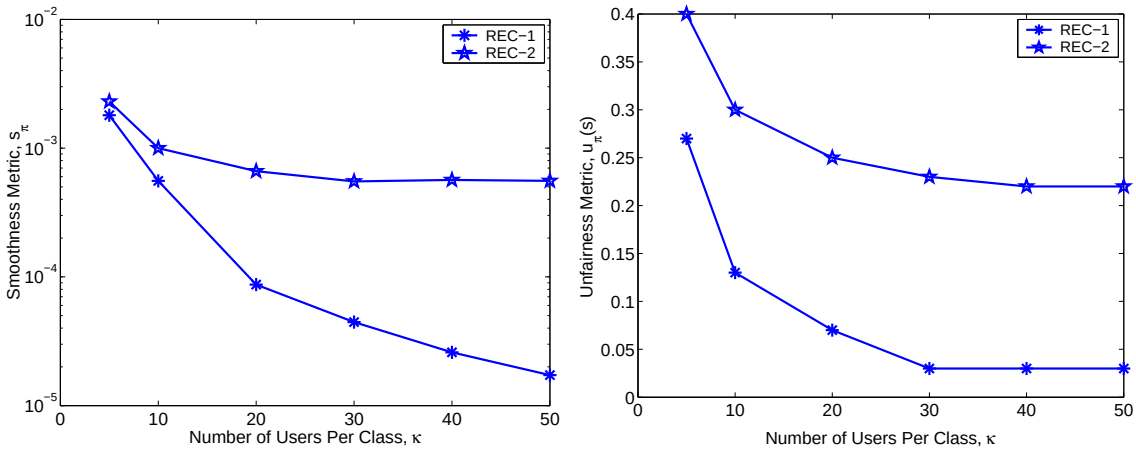

Figure 5: Comparison of s_π (left) and u_π (right) amongst variants of recursive loop schedulers for $\underline{\tilde{r}}$ = [32,75,125] in 3-class scheduling.

Metric	$s_{(REC-1)}/s_{(REC-2)}$						$u_{REC-1}(s)/u_{REC-2}(s)$					
κ	5	10	20	30	40	50	5	10	20	30	40	50
[75,100,125]	1											
[32,75,100]	0.6000	0.3749	0.1240	0.0505	0.0207	0.0111	0.6250	0.2857	0.1579	0.1667	0.0938	0.0682
[32,75,125]	0.7826	0.5571	0.1315	0.0808	0.0458	0.0310	0.6667	0.4444	0.2667	0.1429	0.1538	0.1212
[32,100,125]	1.0000	0.9401	0.5698	0.6098	0.3012	0.2306	0.5714	0.4000	0.2000	0.1739	0.1200	0.0938

Table 2: Table of $\frac{s_{REC-1}}{s_{REC-2}}$ and $\frac{u_{REC-1}(s)}{u_{REC-2}(s)}$ for various 3-class scheduling scenarios.

5.2 Performance Comparison between Class-aware and Class-unaware Loop Schedulers

Next, we compare the smoothness and fairness performance in terms of s_π and m_π between the 'best' variant of recursive loop scheduler (according to the results in Section 5.1 and denoted by REC_K) and the class-unaware loop schedulers defined in Section 3.

Our computations show that the performance of the variants of Round-Robin schedulers (SRR_K, *WRR*-sp^*_K and CRR_K) are similar (within 14% over all the user configurations, κ = {1,5,10,20}, for s_π). Hence, we define a representative Round-Robin scheduler (denoted by $*RR_K$) with performance metric, σ_{*RR_K}, defined as follows, where $\sigma \in \{s, m\}$:

$$\sigma_{*RR_K} = \frac{1}{3}[\sigma_{SRR_K} + \sigma_{WRR-sp^*_K} + \sigma_{CRR_K}]$$

In addition, the RND_K scheduler performs significantly worse than the deterministic schedulers (excluding the DRR_K scheduler) in terms of smoothness, and this performance gap widens as the number of users per class, κ, increases. To quantify this, we evaluate

$\Delta^{max}_{RND_K} = \frac{\min_{\pi \in \{REC_K, MBAL_K, STF_K, *RR_K, GR_K\}} s_\pi}{s_{RND_K}}$ and $\Delta^{min}_{RND_K} = \frac{\max_{\pi \in \{REC_K, MBAL_K, STF_K, *RR_K, GR_K\}} s_\pi}{s_{RND_K}}$ and the results are shown in Table 3. We observe that the performance gap between the RND_K scheduler and the deterministic schedulers widens as the number of users per class, κ, increases. However, even for κ = 1, the worst-case smoothness metric obtained with deterministic schedulers is less than 40% of the corresponding metric obtained with RND_K schedulers. Hence, for easier comparison of the relative performance of the remaining schedulers, both the RND_K and DRR_K schedulers have been deliberately omitted from the figures.

5.2.1 Allocation Smoothness

The results for allocation smoothness for the 4-class scheduling scenario are plotted in Fig. 6. The corresponding results for $\tilde{r}$ = [32,75,125] in 3-class scheduling is shown in Fig. 7.

We observe that the schedulers can be ranked based on their relative smoothness performance (beginning with the best smoothness) as follows: $\{REC_K, MBAL_K, STF_K, *RR_K, GR_K\}$, and such a ranking is consistent in terms of both smoothness metrics and for both scheduling scenarios. This shows that there is some equivalence between the variance-based smoothness metric we proposed, and the notion of balanceness.

In addition, we note that due to its class-awareness, our proposed recursive scheduler gains in smoothness as the user population, κ, increases, while the converse is true for the other class-unaware schedulers. Hence, the recursive scheduler ensures *stability* under high load conditions.

Since the GR_K scheduler exhibits the worst smoothness performance for $\tilde{r}$ = [32,75,125], we tabulate the metrics, $\frac{s_\pi}{s_{GR}}$ and $\frac{m_\pi}{m_{GR}}$, $\pi \in \{*RR_K, STF_K, MBAL_K, REC_K\}$, for each of the remaining 3-class scheduling scenarios in Table 4. We show that the above observations are consistent over all the 3-class scheduling scenarios.

5.2.2 Unfairness

The results for unfairness for the 4-class scheduling scenario are plotted in Fig. 9. The corresponding results for $\tilde{r}$ = [32,75,125] in 3-class scheduling is shown in Fig. 10.

According to Fig. 9 and 10, if we categorize the schedulers into two groups: **A**=$\{MBAL_K, GR_K\}$ and **B**=$\{*RR_K, STF_K, REC_K\}$, then we notice that the Group **B** schedulers achieve better fairness than Group **A** schedulers in terms of both variance-based and balance-based metrics. Hence, as with allocation smoothness, there is some equivalence between the variance-based smoothness metric we proposed, and the notion of balancedness.

Since the $MBAL_K$ scheduler exhibits the worst fairness performance for $\tilde{r}$ = [32,75,125], we tabulate the metrics, $\frac{u_\pi(s)}{u_{MBAL}(s)}$ and $\frac{u_\pi(m)}{u_{MBAL}(m)}$, $\pi \in \{*RR_K, STF_K, GR_K, REC_K\}$, for each of the remaining 3-class scheduling scenarios in Table 5. We show that the above observations are consistent over all the 3-class scheduling scenarios.

Overall, the superior smoothness performance achieved with the $MBAL_K$ scheduler is traded off with unfairness and computational complexity (due to *ITER*) compared with the $*RR_K$ and STF_K schedulers. In addition, while the GR_K scheduler performs poorly in terms of smoothness and fairness, our proposed recursive scheduler achieves the best smoothness with almost no degradation in fairness.

5.3 Performance Comparison with GPS and PGPS schedulers

The Generalized Processor Sharing (GPS) [5] scheduler is an *idealized* scheduler where multiple users are served *simultaneously* and the traffic is assumed to be infinitely divisible (bits as opposed to packets).

In our context, users are assumed to be continuously backlogged and with GPS scheduling, each user *j* with relative demand x^j will be continuously served at a *constant* rate x^j. As a result, the GPS scheduler exhibits ideal smoothness and fairness, i.e., $s_{GPS} = m_{GPS} = 0$ and $u_{GPS} = \min_{\pi \in \mathbf{F}^\Sigma} u_\pi$.

If F_p is the time at which packet *p* will depart (finish service) under GPS, then a very good approximation of GPS would be a work-conserving scheme that serves packets in increasing order of F_p. These packetized implementations of GPS are known as Packetized GPS (or PGPS) [5] or Weighted-Fair Queueing [4]. Under the assumption of continuously-backlogged users and constant packet (slot) size, it can be shown that PGPS schemes are equivalent to the *WRR-*sp_K scheduler.

6 Conclusions

In this paper, we consider the design of a perfectly-fair loop scheduler, where the time-slot assignment amongst *K* users is weighted according to the relative bandwidth requirement, is periodic and is as evenly-spaced (smooth) with respect to each user as possible. Such schedules are useful for QoS provisioning in cellular-type wireless networks and for data forwarding in wireless sensor networks. We consider a class-based scheduling scenario where users are grouped according to their relative bandwidth demands. In addition to (absolute) allocation smoothness, it is desirable for the schedule to ensure intra-class fairness, where users in the same class have the same allocation smoothness.

While the allocation smoothness has been quantified based on the concept of balancedness in existing literature, we propose an alternative smoothness metric based

κ	1		5		10		20.00	
Δ_{RND}	Δ^{max}_{RND}	Δ^{min}_{RND}	Δ^{max}_{RND}	Δ^{min}_{RND}	Δ^{max}_{RND}	Δ^{min}_{RND}	Δ^{max}_{RND}	Δ^{min}_{RND}
[75,100,125]	0.092	0.184	0.001	0.126	0.000	0.119	0.000	0.117
[32,75,100]	0.100	0.274	0.003	0.158	0.001	0.156	0.000	0.156
[32,75,125]	0.116	0.254	0.002	0.148	0.001	0.142	0.000	0.139
[32,100,125]	0.104	0.195	0.002	0.158	0.001	0.154	0.000	0.150
[32,75,100,125]	0.139	0.386	0.002	0.251	0.001	0.246	0.000	0.241

Table 3: Performance gap in terms of smoothness between RND_K and the deterministic schedulers in 3-class scheduling.

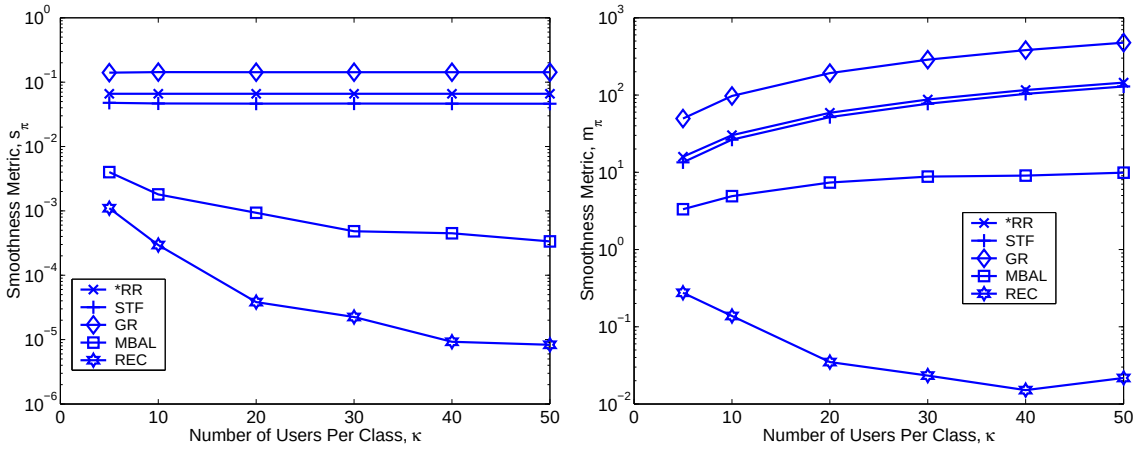

Figure 6: Smoothness Performance of various $\pi \in \mathbf{F}^{\underline{r}}$ for $\underline{\tilde{r}}$ = [32,75,100,125] in 4-class scheduling in terms of s_π (left) and m_π (right).

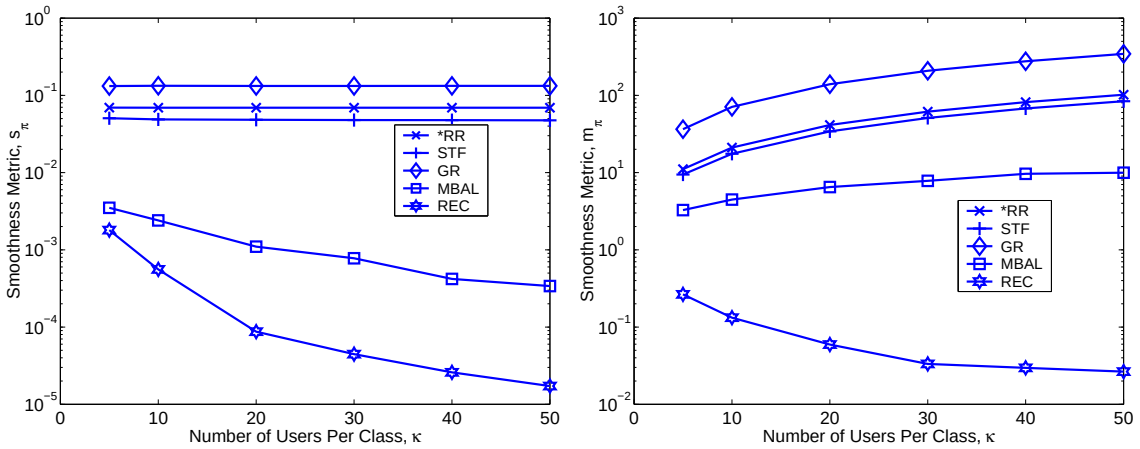

Figure 7: Smoothness Performance of various $\pi \in \mathbf{F}^{\underline{r}}$ for $\underline{\tilde{r}}$ = [32,75,125] in 3-class scheduling in terms of s_π (left) and m_π (right).

on the second moment of the inter-allocation distance for each user, which is more intuitive and also easier to compute. We analyze the allocation smoothness of a weighted round robin with spreading (*WRR-sp_K*) scheduler for a two-class scenario. Based on these properties, we construct an optimal scheduler that employs the *WRR-sp_K* scheduler as an inter-class scheduler, and also suggest an enhancement to the *WRR-sp_K* scheduler. We then propose a recursive class-aware scheduler based on the two-class optimal scheduler for a multiple-class scenario.

We then compare the performance of the above schedulers with other existing loop schedulers. Our proposed scheduler achieves significantly superior smoothness performance with almost no degradation in intra-class fairness. This highlights the importance of class-awareness in the scheduler design for class-based scenarios. In addition, we also demonstrate the equivalence between our proposed metric and the existing smoothness measure based on balancedness, since the relative performance of the schedulers is similar under both types of metrics.

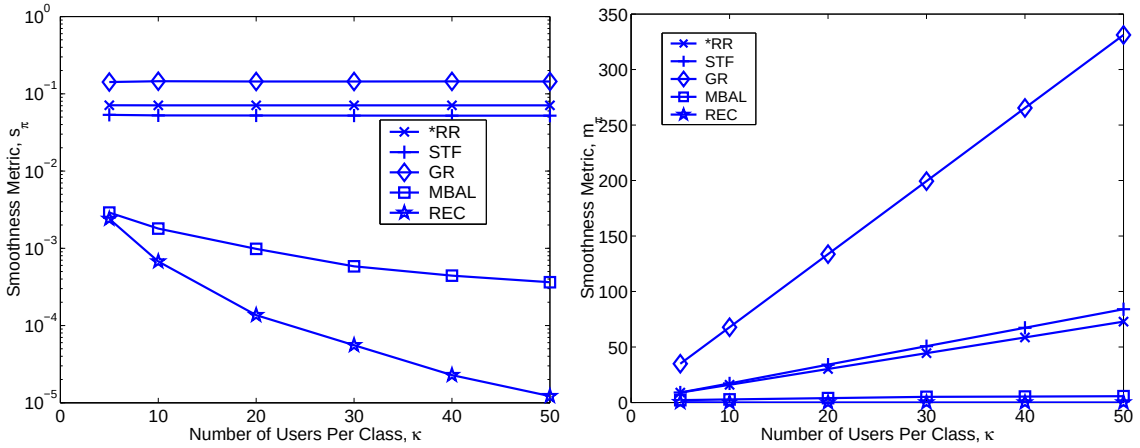

Figure 8: Smoothness Performance of various $\pi \in \mathbf{F}^{\underline{r}}$ for $\underline{\tilde{r}}$ = [32,75,100] in 3-class scheduling in terms of s_π (left) and m_π (right).

[75,100,125]	s_π/s_{GR}						m_π/m_{GR}					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.9820	0.9962	0.9947	0.7725	0.7725	0.9962	1.2463	1.1845	1.1547	0.8871	0.8896	1.1475
STF	0.8065	0.8182	0.8170	0.8182	0.8182	0.8182	0.7105	0.7143	0.7170	0.7217	0.7238	0.7255
MBAL	0.0398	0.0202	1.0000	0.0062	0.0048	0.0038	0.1723	0.1246	0.0808	0.0747	0.0747	0.0602
REC	0.0059	0.0015	0.0000	0.0000	0.0000	0.0001	0.0533	0.0268	0.0000	0.0000	0.0000	0.0054

[32,75,100]	s_π/s_{GR}						m_π/m_{GR}					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.5004	0.4866	0.4913	0.4903	0.4900	0.4903	0.2555	0.2356	0.2259	0.2227	0.2210	0.2200
STF	0.3763	0.3608	0.3629	0.3615	0.3606	0.3608	0.2579	0.2538	0.2549	0.2547	0.2538	0.2538
MBAL	0.0204	0.0124	1.0000	0.0040	0.0031	0.0025	0.0609	0.0409	0.0292	0.0255	0.0202	0.0170
REC	0.0169	0.0046	0.0009	0.0004	0.0002	0.0046	0.0223	0.0023	0.0003	0.0003	0.0001	0.0000

[32,100,125]	s_π/s_{GR}						m_π/m_{GR}					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.4506	0.4427	0.4446	0.4443	0.4440	0.4443	0.2634	0.2557	0.2535	0.2527	0.2521	0.2518
STF	0.3427	0.3345	0.3345	0.3357	0.3340	0.3343	0.2642	0.2677	0.2714	0.2657	0.2688	0.2658
MBAL	0.0310	0.0125	1.0000	0.0034	0.0030	0.0028	0.0544	0.0453	0.0307	0.0253	0.0234	0.0184
REC	0.0148	0.0040	0.0005	0.0005	0.0001	0.0001	0.0098	0.0018	0.0003	0.0002	0.0001	0.0000

Table 4: Table of $\frac{s_\pi}{s_{GR}}$ and $\frac{m_\pi}{m_{GR}}$, for various loop schedulers in 3-class scheduling.

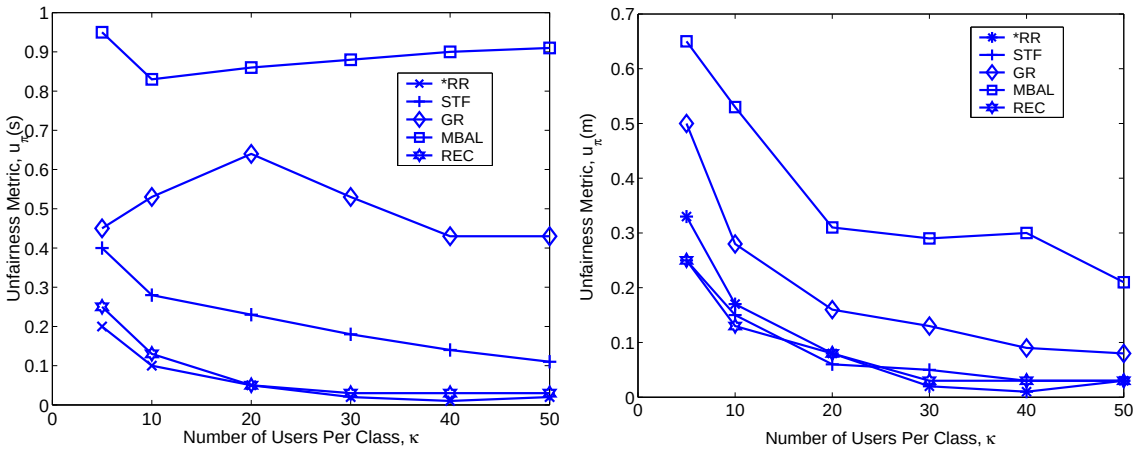

Figure 9: Fairness Performance of various $\pi \in \mathbf{F}^{\underline{r}}$ for $\underline{\tilde{r}}$ = [32,75,100,125] in 4-class scheduling in terms of s_π (left) and m_π (right).

References

[1] L. Kleinrock, "Time-shared Systems: A Theoretical Treatment," *Journal of the ACM*, vol. 14, no. 2, pp. 242–261, April 1967.

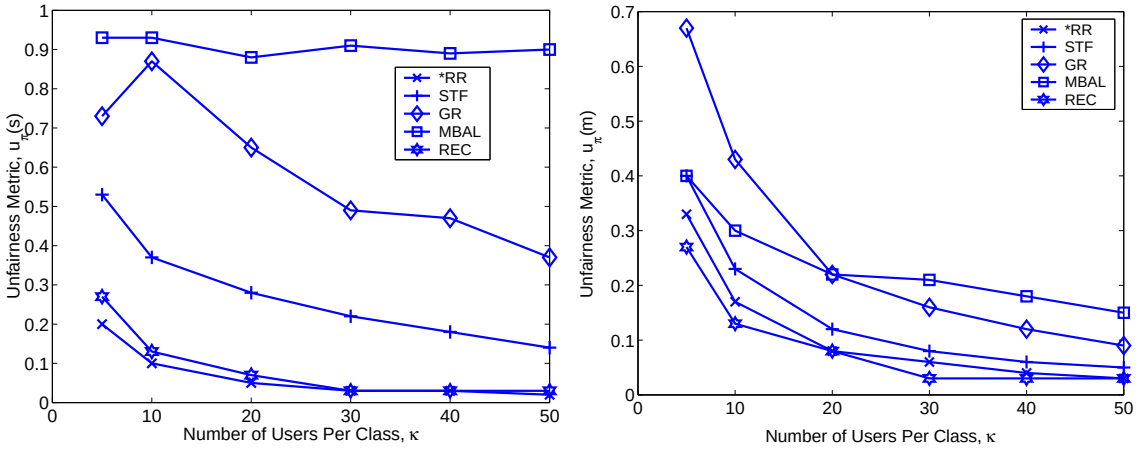

Figure 10: Fairness Performance of various $\pi \in \mathbf{F}^{\underline{r}}$ for $\underline{r}$ = [32,75,125] in 3-class scheduling in terms of s_π (left) and m_π (right).

[75,100,125]	$u_\pi(s)/u_{MBAL}(s)$						$u_\pi(m)/u_{MBAL}(m)$					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.2727	0.1500	0.0833	0.0526	0.0435	0.0345	0.3750	0.2308	0.1200	0.0938	0.0882	0.1619
STF	0.2727	0.1500	0.0833	0.0526	0.0435	0.0345	0.3750	0.2308	0.1200	0.0938	0.0882	0.0857
GR	1.3636	1.5000	1.6667	1.5789	1.7391	1.7241	0.7500	0.5385	0.2400	0.2500	0.2059	0.1429
REC	0.2727	0.1500	0.0833	0.0526	0.0435	0.0345	0.3750	0.3077	0.1200	0.0938	0.0882	0.1143

[32,75,100]	$u_\pi(s)/u_{MBAL}(s)$						$u_\pi(m)/u_{MBAL}(m)$					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.3000	0.1250	0.0732	0.0441	0.0236	0.0638	0.8095	0.6296	0.4722	0.4048	0.3148	0.2833
STF	0.4000	0.1667	0.0976	0.0588	0.0315	0.1064	0.7143	0.4444	0.4167	0.3571	0.2222	0.2000
GR	1.5000	1.2500	1.4634	1.3235	1.1811	1.7021	1.2857	1.0000	1.0833	0.9286	0.6111	0.4500
REC	0.5000	0.1667	0.0732	0.0319	0.0236	0.0851	0.7143	0.5556	0.4167	0.3571	0.2778	0.2000

[32,100,125]	$u_\pi(s)/u_{MBAL}(s)$						$u_\pi(m)/u_{MBAL}(m)$					
κ	5	10	20	30	40	50	5	10	20	30	40	50
*RR	0.2143	0.1111	0.0577	0.0462	0.0313	0.0236	0.6667	0.5185	0.3889	0.2917	0.2745	0.2456
STF	0.2857	0.1481	0.0962	0.0769	0.0521	0.0394	0.8571	0.6667	0.5000	0.3750	0.3529	0.3158
GR	1.0714	1.1111	1.1538	1.3846	1.2500	1.1811	1.4286	1.4444	1.0833	0.9375	0.8235	0.7895
REC	0.2857	0.1481	0.0577	0.0615	0.0313	0.0236	0.7143	0.6667	0.5000	0.3750	0.3529	0.2632

Table 5: Table of $\frac{u_\pi(s)}{u_{MBAL}(s)}$ and $\frac{u_\pi(m)}{u_{MBAL}(m)}$, for various loop schedulers in 3-class scheduling.

[2] J. W. Cohen, "The Multiple Phase Service Network with Generalized Processor Sharing," *Acta Informatica*, vol. 12, pp. 245–284, 1979.

[3] D. Raz, H. Levy, and B. Avi-Itzhak, "A Resource-Allocation Queueing Fairness Measure," *Proceedings of SIGMETRIC/Performance 2004*, pp. 130–141, June 2004.

[4] A. Demers, S. Keshev, and S. Shenker, "Analysis and simulation of a Fair Queueing Algorithm," *Journal of Internetworking: Research and Experiance*, vol. 1, pp. 3–26, October 1990.

[5] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks - the Single Node Case," *IEEE/ACM Transanctions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

[6] S. Lu, V. Bharghavan, and R. Srikant, "Fair scheduling in wireless packet networks," *Proc. of the ACM SIGCOMM*, pp. 63–74, August 1997.

[7] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," *Proc. of the IEEE International Conference on Real-Time Computing Systems and Applications*, pp. 62–69, December 1999.

[8] A. Bar-Noy, R. Bhatia, J. Naor, and B. Schieber, "Minimizing Service and Operation Cost of Periodic Scheduling," *Proc. of the ACM Symposium on Discrete Algorithms*, pp. 11–20, January 1998.

[9] Z. Brakerski, A. Nisgav, and B. Patt-Shamir, "General Perfectly Periodic Scheduling," *Proc. of the ACM Symposium on Principles of Distributed Computing*, pp. 163–172, July 2002.

[10] A. Bar-Noy, A. Nisgav, and B. Patt-Shamir, "Nearly Optimal Perfectly-Periodic Schedules," *Proc. of the*

ACM Symposium on Principles of Distributed Computing, pp. 107–116, August 2001.

[11] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir, "Efficient Periodic Scheduling by Trees," *Proc. of the IEEE INFOCOM*, vol. 2, pp. 791–800, June 2002.

[12] Z. Rosberg, "Optimal Decentralized Control in a Multiaccess Channel with Partial Information," *IEEE Transactions on Automatic Control*, vol. 28, no. 2, pp. 187–193, February 1983.

[13] A. Itai and Z. Rosberg, "A Golden Ratio Control Policy for a Multiple-Access Channel," *IEEE Trans. Information Theory*, vol. 33, no. 3, pp. 341–349, May 1987.

[14] M. Hofri and Z. Rosberg, "Packet Delay under the Golden Ratio Weighted TDM Policy in a Multiple-Access Channel," *IEEE Trans. Information Theory*, vol. 33, no. 3, pp. 341–349, May 1987.

[15] B. Hajek, "Extremal Splittings of Point Processes," *Mathematics of Operations Research*, vol. 10, no. 4, pp. 543–556, November 1985.

[16] E. Altman, B. Gaujal, and A. Hordijk, "Balanced Sequences and Optimal Routing," *Journal of the ACM*, vol. 47, no. 4, pp. 752–775, July 2000.

[17] S. Sano, N. Miyoshi, and R. Kataoka, "*m*-Balanced words: A generalization of balanced words," *Theoretical Computer Science*, vol. 314, no. 1-2, pp. 97–120, February 2004.

[18] Y. Guo and H. Chaskar, "Class-based Quality of Service over Air Interfaces in 4G Mobile Networks," *IEEE Communications Magazine*, vol. 40, no. 3, pp. 132–137, March 2002.

[19] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *IEEE/ACM Transanctions on Networking*, vol. 4, no. 3, pp. 375–385, June 1996.

[20] V. Do and K. Yun, "An Efficient Frame-Based Scheduling Algorithm: Credit Round Robin," *Proc. of the IEEE Workshop on HPSR*, pp. 103–110, June 2003.

[21] R. Rom, M. Sidi, and H. P. Tan, "Performance Analysis of a Recursive Cyclic Scheduler for Class-based Scheduling," *Proc. of the* 16^{th} *ITC Specialist Seminar on Performance Evaluation of Wireless and Mobile Systems*, pp. 43–54, August 2004.

[22] C. Guo, "SRR: An *O*(1) Complexity Packet Scheduler for Flows in Multi-Service Packet Networks," *Proc. of the ACM SIGCOMM*, pp. 211–222, August 2001.

[23] H. Zhang, "Service disciplines for guaranteed performance service in packet - switching networks," *Proceeding of the IEEE*, vol. 83, no. 10, pp. 1374–1399, October 1995.

[24] S. Viswanathan, "Future View of Broadband Demand," FCC TAC Meeting, Intel, April 2003, Available at `http://www.fcc.gov/oet/tac/TAC_III_04_17_03/Future_View_of_Broadband_De%mand.ppt`.