Non-Real-Time Content Scheduling Algorithms for Wireless Data Networks

Samrat Ganguly, Mainak Chatterjee, and Rauf Izmailov, *Member*, *IEEE*

Abstract—A substantial portion of the emerging wireless data service consists of non-real-time applications such as content download. The existing mechanisms based on per-packet performance guarantees used mainly for voice and streaming media do not suffice for the elastic nature of non-real-time traffic. For a non-real-time user data services, the key performance measure of interest is the *total download time*. In this paper, we propose a novel scheduling framework for wireless content service. Specifically, we present a two-layer scheduling architecture that combines content-aware scheduling with opportunistic scheduling. In terms of content-awareness, the proposed scheduling policy provides guarantees on the download time of content. In the second stage, the instantaneous channel conditions of different users are exploited in an opportunistic manner so as to maximize the throughput of the system. We define service differentiation in two modes—differential and guaranteed—and provide polynomial time algorithms for both that manipulate the stretch ratio but within allowable limits. Extensive simulations are conducted that verify the efficiency of the proposed schemes and provide insights into the behavior of the scheduling algorithms for non-real-time data.

Index Terms—Service differentiation, stretch, non-real-time traffic, optimization, deadline scheduling.

1 Introduction

Delivery of *non-real*-time content (elastic applications with no per-packet delay constraints) promises to be a significant revenue earner for future generation wireless networks [10]. Such services include file downloads, Web browsing, interactive games, streaming audio and video, maps, etc. To cater to future demand for data services, the emphasis for most of the existing technologies has been to increase the data rates. For the currently deployed third generation (3G) cellular systems, new technologies are being standardized that support the high data rates required by the data applications. Among the two most popular are *High Speed Downlink Packet Access* (HSDPA) [20] and *High Data Rate* (HDR) [6] proposed by 3GPP [1] and 3GPP2 [2], respectively, both of which are designed for the downlink channels where a time-divisioned-based strategy is employed for allocating channels to the mobile terminals. The key components of both systems are the scheduling policies, the goal of which is to maintain good user performance and overall system throughput. To that extent, opportunistic scheduling strategies were proposed in [8], [16], [17], [19], [24] where, in each time slot, the user with the best channel condition is selected for transmission. Opportunistic scheduling strategy exploits multiuser diversity and maximizes the overall throughput, which might be unfair to users with low long-term signal to interference noise ratio (SINR). To strike a balance between throughput and fairness, proportional fair (PF) scheduling was proposed in [13]. Many other scheduling policies have been proposed and analyzed (see [8] and references therein). Most of these scheduling policies are sensitive to flow-level dynamics. For example, in a PF scheduler, a user is selected based on his current average throughput [13]. However, to a user, it is the *total download time* of a content that matters the most and not the average throughput.

This observation motivates us to design scheduling policies in conjunction with the existing policies that aim to deliver all the data of a given content within an appropriate time based on fairness or quality of service (QoS) objectives. (The word "content" is used both in the singular and plural forms.) Typically, a user does not care about the speed at which he receives the initial portion or the initial bits of the data requested. Therefore, we define the QoS objectives as a generalized service model that classifies the non-real-time content in terms of *stretch factor* [7]. The stretch factor for content is defined in terms of the increase in the download time of content in a loaded system with respect to the download time in an ideal situation where the entire capacity is allocated to that content. This notion also allows us to specify and treat every content with a different QoS level. We define a QoS class for each content based on the stretch ratio that the content might undergo. Since stretch factor was introduced in [7], a variety of scheduling algorithms have been proposed based on this metric, such as minimizing the average or maximum stretch. This metric has also been used in the context of scheduling Web servers [23], [25] and scheduling in CDMA networks [5], [14]. However, to the best of our knowledge, the problem of providing class-based stretch differentiation for emerging wireless data networks has not been addressed so far.

From a content-awareness viewpoint, we set the QoS objectives in terms of the download time. We try to maximize the throughput given that the QoS objectives (deadlines) are met. The goal of the proposed service architecture is to differentiate the content belonging to

- *S. Ganguly and R. Izmailov are with NEC Laboratories America, 4 Independence Way, Princeton, NJ 08540. E-mail: {samrat, rauf}@nec-labs.com.*
- *M. Chatterjee is with the Department of Electrical Engineering and Computer Science, University of Central Florida, PO Box 162450, Orlando, FL 32816-2450. E-mail: mainak@cpe.ucf.edu.*

Manuscript received 7 Mar. 2005; revised 30 Dec. 2005; accepted 24 Feb. 2006; published online 22 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0070-0305.

different classes based on stretch ratio. We propose a content scheduler that can be located at the base station (e.g., a proxy server at the base station) that determines how the different *content* should be pushed to the already existing packet scheduler that provides fairness in terms of achieved bandwidth. Instead of modifying the packet scheduler, we use the content scheduler to manipulate the elastic nature of the content so that the per content QoS objective is met. To that extent, we propose scheduling algorithms for the content scheduler that assign deadlines to each content while maintaining the stretch ratio. Based on the appropriate deadlines, content is simply pushed in the earliest deadline first (EDF) manner to the packet scheduler. Thus, we have two scheduling stages at the base station—content-based and packet-based. In proposing this content scheduling framework, we specifically make the following contributions in this paper:

- We propose a new service model for the non-real-time content delivery based on download time. In this service model, we classify non-real-time content based on the stretch factor.
- In order to maximize the throughput under specific QoS parameters, we provide a two-stage scheduling framework. The first-level scheduler is based on a modified earliest deadline first (EDF) policy that provides the stretch-based QoS. We show that the EDF scheduling can lead to low throughput unless content is pushed simultaneously to the base station in order to accommodate the short term SINR changes.
- At the second stage packet scheduler, we use data partitioning to create parallel schedules from the sequential (EDF) schedules to improve the overall throughput. The parallel schedule is similar to a packet-based opportunistic (C/I) scheduler [20].
- We provide scheduling algorithms that can provide both differential as well as guaranteed service to content based on the stretch ratio.
- We show that if the content is partitioned into equal sized chunks and scheduling is done on these chunks, QoS differentiation can be met, which also results in the increase in the overall system throughput. We provide an algorithm with polynomial time complexity to achieve such partition-based scheduling.
- We conduct extensive simulation experiments to test the performance of the proposed algorithms. A scheduling architecture along with the algorithms was implemented using the event-based simulator OMNeT [22]. The specific goal of the simulation-based evaluation is to show the efficacy of the scheduling strategies in realizing the target QoS on stretch factor and the overall channel utilization.

The rest of the paper is organized as follows: In Section 2, we present the service model with respect to the stretch factor and describe the two-stage scheduling architecture. In Section 3, we propose the scheduling algorithms that provide per content QoS based on stretch ratio. In Section 4, we present the content partitioning algorithm to achieve block level scheduling for increasing utilization. In Section 5, we provide the results of extensive simulations which illustrate the behavior of the scheduling algorithms and their efficiency. Conclusions are drawn in the last section.

2 Service Model and Scheduling Framework

Our objective is to formalize scheduling algorithms for non-real-time content that tries to minimize the download time of the *entire* content. We assume that the content size is known a priori because, for most applications like FTP and HTTP, the content size is known. This research does not deal with streaming media where the content size might not be known in advance.

We consider a single cell cellular system where there is a single base station transmitting data to multiple active users. If the total demand of the users exceeds the transmission capacity of the base station, then the base station queues the data in its buffer. It can be noted that the instantaneous total transmission capacity depends on the channel conditions experienced by the users.

2.1 Download Time and Stretch Factor

Let us consider a content C_p which is admitted into the system. Let a_p denote the *origination time*—the time at which it was admitted. Let f_p denote the finish time (time at which the entire content is delivered at the user equipment). Then, the *download time* is simply obtained by their difference, i.e., $f_p - a_p$. It can be noted that download time is the time taken for the entire content to reach the receiver, i.e., the time difference between when the request for the content was sent and when the last bit of the content was received. Under ideal conditions (when there is no other content queued in the buffer), let the download time be $I(C_p)$. However, in a realistic situation, there is some load due to multiple requests for content; let $R(C_p)$ be the actual download time. The *stretch factor* of content C_p is defined as $\tau(C_p) = R(C_p)/I(C_p)$. The stretch factor shows how the delivery of a given content is slowed due to the presence of other content in buffer and channel conditions. Based on the stretch factor, we define two types of QoS models:

1. *Guaranteed service on stretch.* In this model, QoS classes for each content are defined based on the bound on the stretch factor. In other words, if content C belongs to the class k, it will have a stretch factor $\tau(C) < \tau_k$, where τ_k is the bound on the stretch factor for class k.
2. *Differential service on stretch.* In this model, QoS classes are defined in a relative fashion, i.e., the ratio of the stretch factors between two classes is considered. For example, let k and $k+1$ denote two QoS classes where class k has a higher QoS level than class $k+1$. Let $\tau_k(C)$ and $\tau_{k+1}(C)$ denote the stretch factors of content C belonging to class k and $k+1$, respectively. The class with a higher level has a smaller stretch factor. This is because content belonging to a higher class will be given more priority and the additional delay will be less, thus the stretch factor will be smaller. Thus, we have a ratio of the stretch factor between adjacent classes as $t_k = \tau_k(C)/\tau_{k-1}(C)$, where $t_k < 1$, for $2 \le k \le K$, where K is the total number of QoS classes. In other words, content belonging to a higher class has

TABLE 1
Notations Used

Notations		Meanings
C_i	:	Content i
a_i	:	Arrival time of content C_i
f_i	:	Finishing time of content C_i
$I(i)$	:	Download time of content i under ideal condition
$R(i)$	:	Download time of content i under real condition
K	:	Total number of classes
$\kappa(i)$	:	Class of content i
$\tau(C)$	:	stretch factor of content C
$t_k = \frac{\tau_k(C)}{\tau_{k-1}(C)}$	:	Ratio of stretch factor $\tau_k(C)$ and $\tau_{k-1}(C)$
T_i/T_j	:	Target ratio between stretch factor of class i and j
X_i	:	Original size of content C_i
$S_i = \frac{X_i}{1-p}$	:	Effective size of content C_i
S_i'	:	Stretched effective size of content C_i
b	:	Current bandwidth from all channels

proportionally smaller stretch compared to a lower class [9].

2.2 Terminology

Before we proceed further, let us introduce some of the notations that would be frequently used. For quick reference, the notations can be found in Table 1. Let C_i denote the ith active content that is being downloaded by a user (active content denotes the content that have nonzero bytes remaining in the buffer at the base station). Suppose there are K QoS classes and let $\kappa(i) \rightarrow [1, \ldots, K]$ be the class of the content C_i. Let $T_k = \Pi_{j=1}^{k} t_j$; then, for any two classes l and m with $l \geq m$, $\tau_l/\tau_m = T_l/T_m$.

Let a_i be the arrival time and S_i be the effective size of the content C_i. Effective size is defined as $S_i = X_i/(1-p)$, where p is the frame error rate and X_i is the actual size of the content. Let b_j be the system specified bandwidth for channel j with b representing the total bandwidth of all channels.

2.3 Two-Stage Scheduling Architecture

In order to provide content-level QoS, we propose a two-stage scheduling architecture as shown in Fig. 1. In the first stage, we have the *content deadline scheduler*. The content deadline scheduler can be realized at the base station just by having an additional proxy server for content buffering and scheduling. The content scheduler schedules the overall dispatch of content in a work-conserving fashion. The content scheduler serves the per session virtual buffer, i.e., the data for each session get queued up in the per session buffer. The content scheduler works in the following way: Based on the content scheduling criteria to be met, each content is given a deadline on the finishing time. Based on the deadlines, the content scheduler selects the content to be served at a given time. Once the content is selected, the data for that content is pushed to the second stage, i.e., to the *packet scheduler*. The packet scheduler schedules the packets to the mobile user as per the C/I scheduling policy [20]. Any packet-level scheduling decision made does not affect the content-level QoS. In order to perform the content-level scheduling at the base station, that instantaneous frame error rate (FER) experienced by each user must be made available which is estimated from the buffer occupancy at the packet scheduler. We assume that the FER can be calculated by noting the difference in the number of frames transmitted and the number of frames (after correct decoding) that actually occupy the receiver buffer.

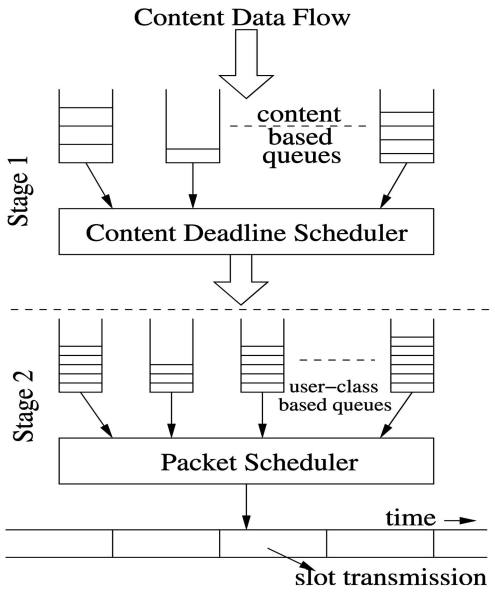

Fig. 1. Two-stage content scheduling.

2.4 Content Partitioning for Opportunistic Scheduling

In HSDPA, the carrier to interference ratio (C/I) scheduler directs transmission to the user with the best channel condition, thus maximizing the overall throughput. In order to take advantage of the C/I scheduler, we try to serve multiple users at the same time. For that, we need buffers at the second stage packet scheduler: buffers that would be nonempty for as many users as possible. This is achieved through content partitioning.

Let us illustrate through an example where we consider content $C1$ and $C2$ as shown in Fig. 2. $C1$ and $C2$ are partitioned into equal-sized segments, each of which takes 1 ms (say) to download. We assume that the content scheduler determines the finishing time of $C1$ and $C2$ to be 9 ms and 14 ms, respectively (one segment each from $C1$ and $C2$ until 9 ms, then five segments of $C2$). In that case, the segments can be pushed in parallel from the content scheduler to the packet scheduler, as shown by the bars. For example, bar 1 implies that all segments below need to be delivered before pushing a segment above bar 1. Segment 5 from content $C1$ before bar 2 signifies that the last segment of $C1$ should be pushed before the deadline (9 ms). This implies that any ordering of segments among themselves below bar 1 does not affect the QoS differentiation or guarantee since download time depends upon the time

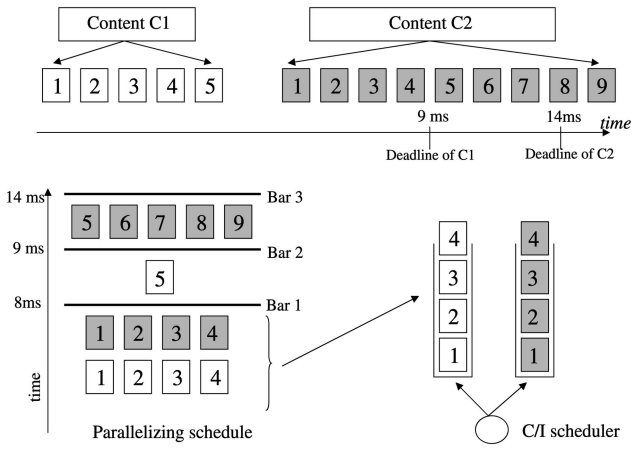

Fig. 2. Content partitioning to maximize throughput.

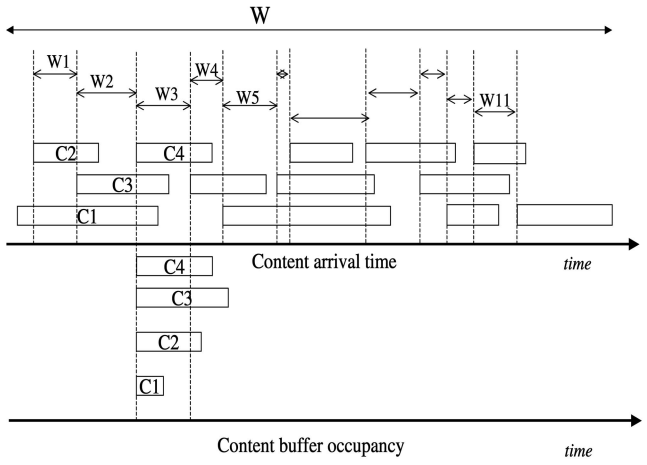

Fig. 3. Content partitioning to maximize throughput.

when the last bit of content is received. Based on this parallel schedule, both queues are occupied simultaneously at the packet scheduler. This makes the C/I scheduler work effectively in maximizing the throughput. Thus, we observe that the proposed scheme tries to maximize throughput with the constraint that the QoS is met.

2.5 Effect of User FER on Scheduling

In order to perform content scheduling, the HS-DSCHs are modeled as a common pool of bandwidth available for sharing. However, the scheduler must capture the different channel quality or frame error rate (FER) of each user and the corresponding perceived bandwidth. This is done by defining the effective content size. Due to channel losses and subsequent retransmissions, the effective bandwidth observed would be $S_{eff} = S/(1-p)$, where S is the original content size and p is the packet loss rate of the channel. (This equality holds true if we do not restrict the number of possible retransmissions.) If the system supports soft-combining, then, instead of discarding corrupted data, the receiver buffers the data and coherently combines them with the received soft information of the retransmission of the bad packet. The error recovery mechanism is initiated by the radio link control which triggers retransmissions to salvage the damaged packets. Note that the FER experienced by a packet during its first transmission is not the same as that during the second transmission. It is smaller in the latter case because there is at least a 3 dB gain in the bit energy-to-noise ratio (E_b/N_o) due to packet combining (assuming that the channel conditions are relatively static between transmission and retransmission) [21]. Let us assume that, due to packet combining, the FER is reduced by a fraction of c. Then, it can be shown that the effective size would be $S_{eff} = S/(1-cp)$.

3 Content Scheduling

In this section, we describe the content scheduling algorithms that try to achieve the target QoS for content delivery.

3.1 Content Scheduling for QoS Differentiation

Based on the allowable stretch for each content, the content scheduler must differentiate the QoS of each content. We define two types of QoS differentiation based on the time period of observation.

- **Instantaneous QoS differentiation.** In this case, the scheduling strategy is based on the pending jobs. These jobs (in the buffer) compete for priority in scheduling based on the class-based stretch ratio. The advantage of such a strategy is that a user has higher flexibility in prioritizing his content download by choosing a higher class and negating the effect of overload. The disadvantage arises from the fact that the user may not be aware of the current buffer size or overload.
- **Long-term QoS differentiation.** Long-term QoS differentiation tries to achieve the stretch ratio based on the history up to the current point, i.e., the content downloaded so far. Scheduling under this strategy helps in differentiating the average effect on the stretch ratio among classes. The advantages and disadvantages of this strategy are just the reverse of those of the instantaneous case.

An Illustrative Example. Fig. 3 shows the arrival of different content. We consider a finite window of observation, W, and partition that into multiple nonoverlapping time windows, as shown by $W1, W2, \cdots W11$. Let us consider the content buffer occupancy during time window $W3$ (bottom part of the figure). In that time window, content C_1, C_2, C_3, and C_4 are considered for scheduling. It is to be noted that the delivery of the bigger content C_1 delays the delivery of C_2, C_3, and C_4. On the other hand, in the long-term case, the instantaneous buffer occupancy is not considered in the scheduling decision. Instead, long term QoS differentiation meets the per class achieved QoS, which is a long term average based on all content that have been served so far.

We consider two cases of scheduling: *nonpartitioned* and *partitioned*. In the nonpartitioned case, scheduling requires content with earliest deadline to be pushed completely to the packet scheduler before the next content can be pushed and is similar to conventional nonpreemptive scheduling. In the partitioned case, we consider that content is partitioned into equal size chunks that are scheduled in a preemptive fashion and, in some regard, is similar to conventional preemptive scheduling. Scheduling for the partitioned case, although it

TABLE 2
Various Scheduling Instances

Schedules	τ_1	τ_2	τ_3	$\frac{\tau_2}{\tau_1}$	$\frac{\tau_3}{\tau_2}$	$\frac{\tau_3}{\tau_1}$	Max	Mean	Max-diff (Δ)
1. $C_1 \rightarrow C_2 \rightarrow C_3$	1.0	1.5	3.0	1.5	2.0	3.0	3.0	1.8	2.0
2. $C_1 \rightarrow C_3 \rightarrow C_2$	1.0	2.3	1.7	2.3	0.7	1.7	2.3	1.6	1.3
3. $C_2 \rightarrow C_1 \rightarrow C_3$	3.0	1.0	3.0	0.3	3.0	1.0	3.0	2.3	2.0
4. $C_2 \rightarrow C_3 \rightarrow C_1$	4.5	1.0	2.9	0.2	2.9	0.6	4.5	2.8	3.5
5. $C_3 \rightarrow C_1 \rightarrow C_2$	2.5	2.2	1.0	0.9	0.5	0.4	2.5	1.9	1.5
6. $C_3 \rightarrow C_2 \rightarrow C_1$	4.5	1.7	1.0	0.4	0.6	0.2	4.5	2.4	3.5

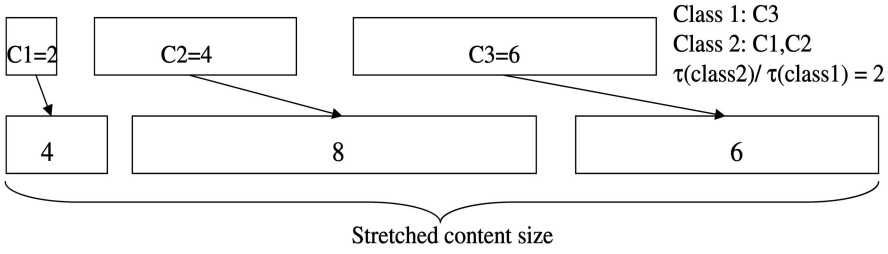

Fig. 4. Content of two classes.

involves higher complexity due to content partitioning, increases throughput, as we discuss next.

3.2 Preliminary Observations

We first consider a simple offline scheduling where, in a given instance of a time window, the size and arrival time of all the content are known. Even with this knowledge, realization of the target stretch ratio with *exact precision* may not be possible. This is because of the different content sizes.

Let us consider an example where there three pieces of content arrive simultaneously: C_1 with download time 2, C_2 with download time 4, and C_3 with download time 3. Also consider that all three content belong to the same QoS class. In the nonpreemptive approach, we need to assign deadlines to these contents and schedule them in earliest deadline first (EDF) fashion. There are six possible schedules, as shown in Table 2. We note that the stretch ratios are different under different schedules. Certainly, it is not equal to $\tau(i)/\tau(j) = 1$ for any schedule, which is our target. Thus, we need to define a relaxed metric that captures how close the stretch ratios are to their target values. For a given schedule, we define *max-diff*, denoted by Δ, as

$$\Delta = \max_{i,j}\left[\frac{\max[T_{\kappa(j)}\tau(i), T_{\kappa(i)}\tau(j)]}{\min[T_{\kappa(j)}\tau(i), T_{\kappa(i)}\tau(j)]} - 1\right] \quad \forall i, j; i \neq j, \qquad (1)$$

where i and j refer to two different contents. Δ is the ratio of a maximum and a minimum value. Since finding the maximum or minimum can be done in $O(n)$ time, the complexity of calculating Δ is linear in time.

This definition of Δ gives the maximum difference between the achieved stretch ratio and target stretch ratio. Class weights T are multiplied to the content stretch to normalize the stretch ratios so that we can compare the ratios with the target value 1. In this definition, we make sure that the stretch ratio is always greater than 1 by dividing *max stretch* by *min stretch*. Therefore, the value of Δ provides the bound on how bad the achieved stretch ratio is. The value of Δ also has implications on the fairness among different content. A high value of Δ means that certain content is unfairly treated compared to others.

Again referring to Table 2, we observe that the value of Δ is minimum for schedule 2. Note that, for the same schedule, the mean stretch and max stretch is minimized. Based on this observation, one can think of a schedule that minimizes the mean stretch or the max stretch for all content. For minimizing mean stretch, the content with the earliest arrival time, a_i, can be scheduled first, which has $O(1)$ competitive ratio [12]. A near optimal algorithm for offline minimization of max stretch is provided in [14] for the preemptive case. In general, minimizing the max stretch is a better indication for minimizing Δ, which we show in the context of multiple QoS classes.

In the case of multiple QoS classes, because of the class weights T_k, scheduling based just on arrival times and sizes will not work. In order to incorporate weights, we stretch the content size such that the stretched content size, $S_i' = T_{\kappa(i)}S_i$ for content C_i. For example, in Fig. 4, we see how the content belonging to two different classes are stretched. Note that stretched content size is only used for finding the schedule, whereas the actual stretch is defined on effective content size S. Consider that content C_1, C_2, and C_3 have the same arriving time. C_1 (size 2) and C_2 (size 4) belong to class 2 and C_3 (size 6) belongs to class 1. Assume $T_1 = 1$ and $T_2 = 2$. Minimizing mean stretch or max stretch with schedule $C_1 \rightarrow C_2 \rightarrow C_3$ based on just effective size gives $\Delta = 3$. Based on stretched content size, minimizing mean stretch with schedule $C_1 \rightarrow C_3 \rightarrow C_2$ also gives $\Delta = 3$. On the other hand, minimizing the max stretch with schedule $C_3 \rightarrow C_1 \rightarrow C_2$ gives $\Delta = 1$. From the above example, we note that minimizing mean stretch may not work in all cases. Instead, we make the following observation:

```
Event: A new content C*:(S*, a*) arrives
Δ_min = some large value
For i = 1 to n+1
Begin
  τ(C*) = (Σ_{j=1}^{i-1} S_j/b + S*/b − a*)/(S*/b)
  τ(C_{k-1}) = (f_{k-1} + S*/b − a_{k-1})/(S_{k-1}/b)
  min_R = min_{k=i+1}^{n+1} (1/T_{κ(C_{k-1})}) τ(C_{k-1})
  max_L = max_{k=1}^{i-1} (1/T_{κ(C_k)}) τ(C_k)
  Δ = max[max_L, (1/T_{κ(C*)})τ(C*)] / min[min_R, (1/T_{κ(C*)})τ(C*)]
  If Δ < Δ_min
    i* = i
    Δ_min = Δ
End
Update f_k for C_k where k ≥ i*
```

Fig. 5. Algorithm for schedules updates.

Observation 1. *Minimizing max stretch based on stretched content size works better in minimizing Δ.*

Unfortunately, minimizing max stretch in a nonpreemptive case is a hard problem [7]. Therefore, we use the above observation for our preemptive (with content partitioned) case, as will be discussed in Section 3.3.2. For the nonpartitioned case, we propose an algorithm that tries to minimize Δ in a sequential manner.

3.3 Instantaneous QoS Differentiation

Based on the observation and metric introduced above, the scheduling problem for instantaneous QoS differentiation can be stated as follows:

Given the sizes, arrival times, and the target stretch ratios of a set of content belonging to multiple classes, find a schedule that minimizes Δ.

We present the algorithms corresponding to the two cases: nonpartitioned and partitioned.

3.3.1 Nonpartitioned Case

Consider a situation where we already have a schedule for which Δ is minimized. Let us now consider the arrival of a new content C^* at time a^* with effective size S^*. Based on this new content, we have to create a new schedule. We do not allow reordering of the data from the same content since data is pushed to the mobile device using a single transport session and not reassembled or reordered at the device. Thus, the problem is to find the proper place of C^* in the existing schedule such that Δ is minimized.

Let the position of C^* be i, $1 \le i \le n$, in the new schedule $C_1 \dots C_n$. Then, the stretch ratios between content with an index less than i will not change. Consider two contents with index l and m such that their indices are greater than i and $\tau(l) > \tau(m)$ in the old schedule. In the new schedule,

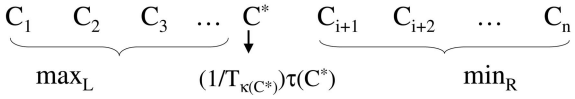

Fig. 6. Finding Δ for a given schedule.

with addition of C^*, the real download time of C_l and C_m will increase by S^*/b, where b is the bandwidth. With this increase, the relative stretch ratio between C_l and C_m can only decrease since

$$\frac{(R(l)+S^*/b)I(m)}{(R(m)+S^*/b)I(l)} \le \frac{R(l)I(m)}{(R(m)I(l)}. \quad (2)$$

However, if we consider the two content with indices l and m such that $m \le i \le l$, then (2) does not hold true. Therefore, the position of i needs to be chosen such that the ratio of the maximum stretch and minimum stretch is minimized. In other words, the index i of C^* is given by

$$i^* = \min_i \left[\frac{\max_l [T_{\kappa(C_m)}(R(l)+S^*/b)/I(l)]}{\min_m [T_{\kappa(C_l)} R(m)I(m)]} \right] \quad \forall l, m; l \ne m. \quad (3)$$

In (3), we find the ratio of max stretch among all content with $l \ge i$ and min stretch among all content with $m \le i$. This ratio, r_{max}, is a function of i and the index of content C^*. We choose i^* such that r_{max} is minimized. The detailed algorithm for updating the schedule on arrival of a new content C^* is given in Fig. 5. Some notations used in the algorithm are clarified next.

In this algorithm, the current schedule that needs to be updated has content $C_1 \dots C_n$ in order. For a given content C_i, let f_i, a_i, and S_i denote the finishing time, arrival time, and effective size, respectively. Due to the work-conserving feature of the schedule $f_{max} = \sum_i S_i/b$, where f_{max} denotes the maximum finishing time. The algorithm finds the index i^* where the content C^* needs to be inserted. In other words, the existing schedule $C_1, \dots, C_n$ becomes $C_1, \dots, C_{i^*}, C_{i^*+1}, \dots, C_{n+1}$.

Let $\max_L$ refer to the maximum of the weighted stretch among the content that are before (L), C^*, as shown in Fig. 6. Similarly, $\min_R$ is the minimum weighted stretch after (R), C^*. In order to find the index i^*, the algorithm finds the maximum Δ for the given index obtained from $\max_L$ and $\min_R$, as shown in Fig. 5. Therefore, in each iteration for a given value of i, the total computation complexity is $O(n \log n)$, resulting in a total complexity of $O(n^2 \log n)$ for the algorithm.

3.3.2 Partitioned Case

In this case, we consider the option that each content can be partitioned into equal size chunks. Since the finishing time of a content refers to the time when the last chunk of the content is downloaded, the problem therefore is to assign the correct finishing time to the last chunk of each content. Based on Observation 1, finishing time f' of content C_i assigned under stretched content size S_i' should minimize the maximum stretch, τ_{max}, with the goal toward minimizing Δ. Thus, the problem can be formally expressed as:

```
1: Check feasibility of f_i with τ_low
2: If feasibility fails: τ_up = 2τ_low
   Check feasibility of f_i with τ_up
   If feasibility fails: τ_low = τ_up
    go to step 1
Else stop
3: Perform a binary search between τ_low and τ_up
stop when difference in consecutive τ values is
less than a threshold.
```

Fig. 7. Algorithm to find minimum max-stretch.

```
Sort f_i in ascending order
PSUM = 0
for i=1 to n
if f_i ≥ PSUM + S'_i/b
   PSUM += S'_i/b
else
   feasibility test failed,
break
```

Fig. 8. Feasibility check in finding minimum max-stretch.

$$Find\ f'_1, f'_2, \cdots, f'_n\ such\ that$$
$$\tau_{max} = \max[\tau(C_1), \ldots, \tau(C_n)]\ is\ minimized.$$

This formulation is based on stretched content size and, so, the finishing time is also stretched. The actual finishing time f_i is derived as

$$f_i = (1/T_{\kappa(C_i)})f'_i. \tag{4}$$

Since the finishing time of the last chunk cannot be less than f_i, which is also true for the finishing times of other chunks belonging to the same content, we have

$$f'_i \geq a_i + S'_i/b. \tag{5}$$

We perform a binary search to find the optimal value of f'_i. To test the feasibility of a given τ, we express the finishing time of each content as $f'_i = a_i + \tau S'_i/b$. We choose τ_{low} and τ_{up} as the initial and final values respectively. We start with a initial value of $\tau_{low} = 1$ and follow the algorithm given in Fig. 7. Since the actual stretch for C_i can be less than τ, the corresponding finishing time is the least possible. The feasibility of the solution requires $f_i > f_j$, implying that all the chunks of C_i and C_j are pushed before f_i. The "threshold" is just a stopping condition that determines the level of coarseness needed for the solution. In order to check the feasibility, we follow the procedure in Fig. 8.

The search for τ_{max} and the feasibility check are done in polynomial time. Since the finishing time is calculated based on the maximum stretch of each content, it might so happen that the maximum finishing time is greater than f_{max}. Therefore, the finishing time should merely be used for ordering and not for pushing or dispatching the content. Content pushing should happen in a work conservative manner. At each instant when schedules are updated, there might be some content which is partially delivered. In the schedule computation, this content is also considered along with its arrival in the same way as other content. Therefore, it is possible that the finishing time of this content could be delayed due to the arrival of new content.

3.4 Long-Term QoS Differentiation

In long-term scheduling, our objective is to minimize Δ based on the content that has already arrived. Since the nonpartitioned case is equivalent to the nonpreemptive approach, it is hard to achieve this goal [7]. Therefore, we consider the partitioned case, which is equivalent to the preemptive approach. The algorithm that we provide is motivated by related work presented in [7], [14]. It is an online algorithm that tries to minimize the maximum stretch on the stretched content size toward minimizing Δ. In this algorithm, as in [7], [14], we maintain an estimate of max-stretch-so-far, τ_{est}.

When new content arrives, the finishing time schedule of each content in the buffer is updated as

$$f'_i \leftarrow a_i + \beta\tau_{est}S'_i/b, \tag{6}$$

where S'_i is the stretched content size as discussed before. The value of β must be greater than 1 [7]. The content is then scheduled based on the finishing times, as done in the instantaneous case. Once the content is completely delivered, the τ_{est} is updated as

$$\tau_{est} \leftarrow (1-\alpha)\tau_{est} + \alpha\tau_{current}, \tag{7}$$

where $\tau_{current}$ is computed based on the stretched size of the content. α controls how much to react to instantaneous change in the max stretch. With time evolution, the algorithm adapts the value of τ_{est} to get near to the minimum max stretch. A detailed algorithm is given in Fig. 9.

Based on the finishing times obtained from the above algorithm, parallel schedules using block formation are used for pushing the content to the packet scheduler.

3.5 Scheduling for Guaranteed Stretch

Let $\tau_k(C)$ be the maximum stretch factor for class k. In order to realize the per class stretch factor, we need a scheduler that creates the deadline for the dispatch of each content in the content buffer. Let $C_1 \ldots C_n$ be the active content that is in the buffer. For content C_i, the maximum stretch factor is $\tau_k(C_i)$, which implies that the maximum allowable deadline is $f_i = a_i + \tau_k(C_i)S_i/b$. Note that the parallel dispatch schedule of the content does not change the total download time as compared to a sequential dispatch schedule. Therefore, we consider a sequential schedule where the contents are sorted based on their maximum allowable deadlines. Using this schedule, the contents are then dispatched to the packet scheduler at the base station. The schedule (order of content) is updated based on the arrival

Event: A new content arrives

For all content C_i in buffer

$S_i' = T_{\kappa(C_i)} S(i)$

$f_i' = a_i + \tau_{est} S_i'/b$

$f_i = (1/T_{\kappa(C_i)}) f_i'$

Schedule the content based on this finishing time

Event: Delivery of content C_i is completed at t

$S_i' = T_{\kappa(C_i)} S_i$

$f_i' = T_{\kappa(C_i)} t$

$\tau_{current} = (f_i' - a_i) b / S_i'$

$\tau_{est} = (1-\alpha)\tau_{est} + \alpha \tau_{current}$

Fig. 9. Algorithm for scheduling in long-term QoS differentiation.

of new content or change in the long term SINR. When new content arrives, its admission feasibility check is done in the following way: We consider the existing content in the buffer along with the newly arrived content. All existing content in the buffer, including the new content with f_j such that $f_j \leq f_i$, can be served before f_i while keeping enough space to serve content C_i by f_i or

$$\sum_{j\, s.t. f_j \leq f_i} S_j/b + S_i/b \leq f_i. \tag{8}$$

This feasibility condition is checked for the deadline of all content including the new content. The same algorithm presented in Fig. 8 can be used for this purpose.

4 Parallelizing Schedule with Partitions

In Fig. 2, we saw how the chunks can be pushed in parallel to the packet scheduler based on the target finishing time of the content. Though it is theoretically possible to emulate the C/I scheduling at the content level, for all practical systems, it is almost impossible to report the momentary changes in user channel states to the content scheduler. Therefore, our goal is to keep maximum number of users queues nonempty at the packet scheduler so that the C/I scheduler can always find a queue to serve corresponding to the best user. In trying to achieve the above strategy, we first group all the content on a per-user basis. Let V_u denote an ordered set of content for user u with finishing times in ascending order.

The parallel schedules are expressed in terms of blocks. Each block consists of one or more chunks from the same or different content. While pushing the content, it is required that block B_i be pushed before B_{i+1}. For better performance, it is necessary that B_i be completely served by the packet scheduler before B_{i+1} can be pushed by the content scheduler. This can be achieved either by monitoring the buffer at the packet scheduler or by estimating the finishing time of B_i based on block size (number of chunks in it) and the bandwidth available. We next show how to construct the blocks.

Let F denote the order set of finishing time for all content, where $F = \{f_1, \ldots, f_i, \ldots, f_n\}$ with $f_i \leq f_{i+1}$ and

Block formation

for $i = 1$ to n

block size = $\lfloor (f_1 \times b/S_\delta) \rfloor - 1$

for u = 1 to v (v: total number of users

$C_p \leftarrow$ = Next(V_u)

check feasibility (p, i, L)

if feasible

$B_m \leftarrow \delta(C_p)$

$\omega(C_p) += -1$

$L = L + 1$

$B_{m+1} \leftarrow \delta(C_i)$

$m = m + 2$

Feasibility check (p, i, L)

for u = i to p-1

if $(f_u - L) < \omega(C_u))$ feasibility fail

else $L += \omega(C_u)$

Fig. 10. Algorithm to form blocks.

$|F| = n$. Let $\omega(C)$ denote the number of chunks that has been assigned to content C. For total content $C_1 \ldots C_n$ indexed by the order in F, we present the block formation algorithm in Fig. 10.

We first find the block size for block B_1. Basically, we check the first deadline to be met given by f_1 and find how many chunks can be pushed by the deadline. We create a separate block B_2 for the last chunk of the content C_1 with finishing time f_1. This is because the last chunk of C_1 cannot be reordered with any chunks, as was shown before in Fig. 2. Given the block size, we compute the chunk assignment. The chunk assignment provides for a given chunk in a block, say, B_1, which belongs to content C. In order to do that, we go in round-robin fashion on each user's V_u. $Next(V_u)$ in Fig. 10 gives the next content C_p in the ordered V which is not covered by any block and whose finishing index $f_p > f_i$. We define $\delta(C_p)$ as a pointer that points to the next chunk in the content C_p that is yet to be assigned to any block and S_δ is the size of that chunk. B_1 is defined in terms of the total number of chunks it contains and is given by $\lfloor (f_1 \times b/S_\delta) \rfloor - 1$. Feasibility is checked on the chunk $\delta(C_p)$ where it is decided if, by inserting the chunk, $\delta(C_p)$ leads to not meeting any future finishing time. If feasible, we add the chunk to the current block B_m and increase L by one, where L denotes the total size of the blocks created so far, i.e., $L = \sum_1^m |B_m|$. Next, the chunk with index f_i is inserted into a separate block B_{m+1}. The process is repeated over set F, yielding a complexity of $O(|N_\delta n^2|)$, where N_δ is the total number of chunks. The maximum total number of blocks formed can be $2n$. The overall scheduling architecture is shown in Fig. 11.

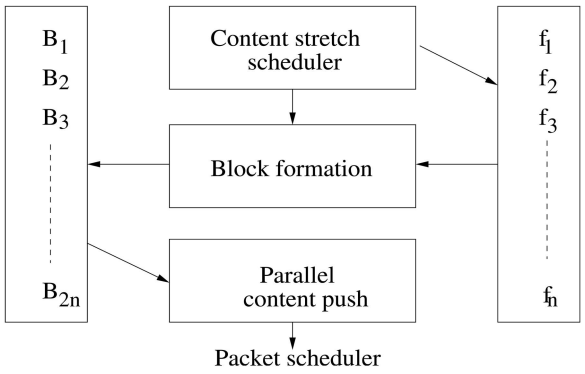

Fig. 11. Block scheduling architecture.

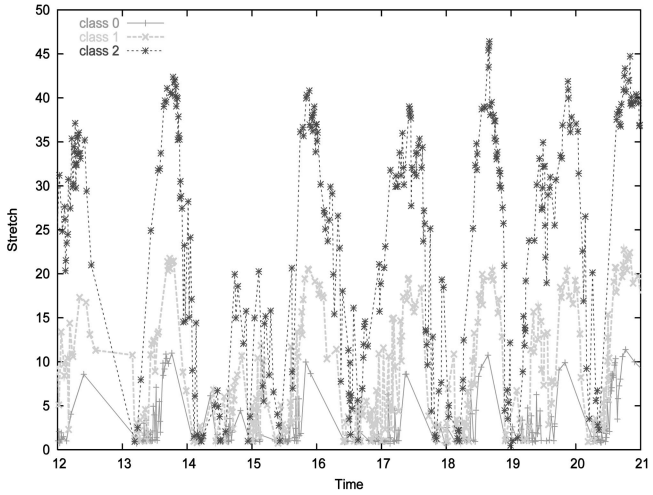

Fig. 12. Stretch for different classes with time.

5 Simulation Model and Results

Implementation of the HSDPA system and content scheduler was done on a discrete event simulation platform provided in OMNeT [22]. In realizing the HSDPA model, we use a simplified analytical model based on the system level and link level simulation suggested in [3]. The system model assumption is a 3-sectored 19-hexagonal cell model with the sector antenna beam pattern based on [3]. Eighty percent of power and 15 codes of spreading factor 16 are allocated for HSDPA service with frame length of one transmission time interval (TTI = 2 msec). Also, we assume hybrid ARQ with chase combining and AMC (automatic modulation and coding), which control the MCS (modulation and coding schemes) according to the average received SINR over one TTI. Sixteen QAM modulation with rate equal to 3/4 is used for peak rate up to 10.8 Mbps. Meanwhile, the location of each user is randomly assigned with a uniform distribution within each cell. The corresponding SINR of each user is fixed in the control interval of packet scheduling that is one TTI. SINR is mapped to nominal user throughput samples using throughput hull curve in analytical simulation. We assumed the pedestrian mobility model. The peak rate of user throughput in the simulation scenario is 10.8 Mbps. The mean rate that can be obtained in one cell is around 5 Mbps, based on channel conditions.

The traffic model is based on the modified ETSI WWW browsing model [4]. We assume that the size of each packet call is a bounded *Pareto* distribution with a minimum size of 4.5 Kbytes and maximum size of 2 Mbytes. Furthermore, the reading time of the content is approximated as a geometric distribution. The content scheduler assumes that the content of a request arrives at the base station at a rate much faster than the individual user's data rate so that the content for a request is available instantly at the base station. This assumption fits with the proxy architecture working with the store and forward mechanism. The content is pushed onto the packet scheduler according to the mechanisms discussed earlier. The packet scheduler works in a work-conserving way—it schedules a frame with a TTI length as long as the packet queue is nonempty. Three types of packet scheduling disciplines are implemented in the simulation platform: round-robin (RR, also called time fair sharing), maximal C/I, and proportional fair sharing.

5.1 Performance Metrics

The three most important performance metrics are stretch, stretch ratio between classes, and overall throughput. The primary goal of the scheduling proposed in our service framework is to achieve the stretch ratio based on content class. The secondary goal is to maximize the overall throughput by utilizing the C/I scheduling to the fullest extent by parallelizing content pushing while meeting the primary goal. In all our experiment scenarios (unless otherwise stated), we consider three classes and a varying number of users, reflecting the total system load. Class 0 is the highest class with the lowest relative stretch and class 2 is the lowest class with the highest relative stretch. The target stretch ratio between class 1 and class 0 is 2 and the target ratio between class 2 and class 1 is also 2. The data traffic for each user is independent of the others. The load is controlled by scaling the load parameters for each user. The SINR for each user is varied independently following the model presented in [15]. We assume that the average frame error rate for each channel is the same but time varying based on changing SINR. The packet scheduler pulls a packet from the packet queues based on backlogged retransmission frames. We do not consider buffer overflow. Instantaneous user SINR is not reported to the content scheduler. Long-term SINR averaged over monitoring a time window of 30 TTI is used in calculating the effective size of the content.

For measuring the performance of the proposed scheduling schemes, we monitor every content that arrives at the content scheduler and the time durations the content stays in the content and packet scheduler. Based on the waiting time at these two scheduler buffers, we obtain the total download time for each content. We organize our simulation study in terms of the evaluation criteria.

5.2 QoS with Guaranteed Stretch

We study the performance of our proposed scheduling framework for six metrics which are relevant for guaranteed stretch.

5.2.1 Transient Stretch for Classes

The evolution of stretch with time for each class is shown in Fig. 12. The figure shows a time window of 10 seconds. We

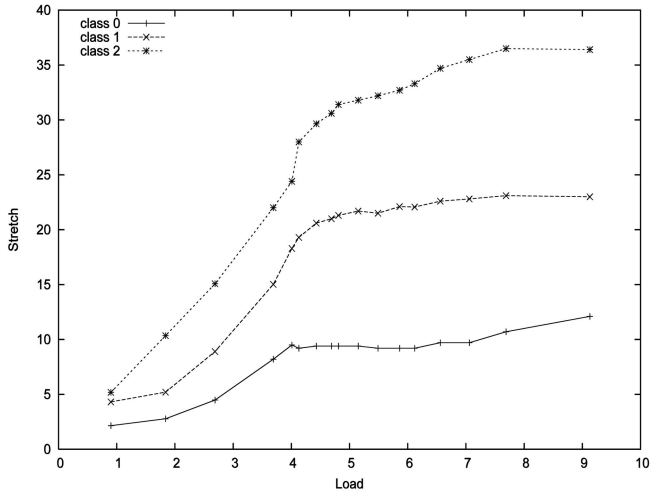

Fig. 13. Maximum stretch for each class versus load.

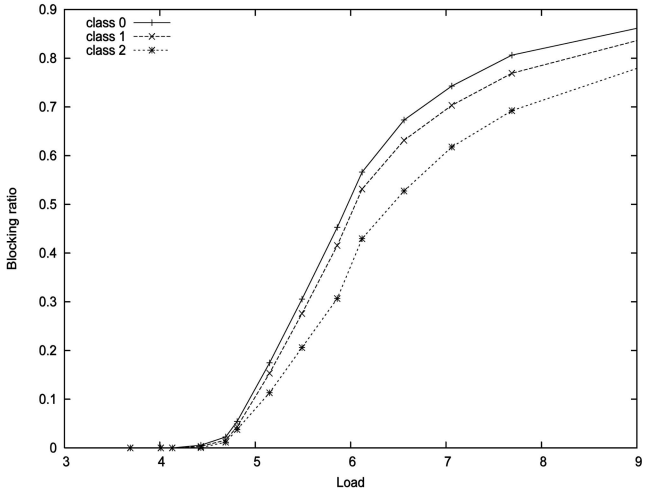

Fig. 14. Blocking ratio of each class versus load.

observe that the stretch factor for each class varies very abruptly with time. This is due to very transient buffer occupancy based on the traffic model assumed. At a certain point when the buffer is underloaded, we note that the stretch factor for each class becomes close to 1. In the transient underloaded scenario, the stretch bound of all classes is naturally met. However, the important point to note here is that, when the buffer becomes full, as represented by points where the stretch value spikes, the bound on stretch is usually met. For example, in the window [13:14] secs, we observe that the max stretch value is around 10, 20, and 43 for classes 0, 1, and 2, respectively. This shows the effectiveness of the admission control and scheduling decision in realizing the stretch bound, even at the transient level.

5.2.2 Max Stretch with Load

Next, our target is to vary the traffic load and observe how the maximum stretch of each class depends on load. For a wide range of load conditions evaluated, we note from Fig. 13 that stretch increases with load and, finally, saturates at the target stretch bound values. This saturation is due to the finite content buffer size; if the buffer is not available to download the entire content, the content is not downloaded at the content scheduler. At low load, stretch values are close to 1, which indicates that the download time of content is not delayed. This in turn implies that the buffer occupancy in terms of the number of content is close to 1.

5.2.3 Blocking Ratio on Load

Since we employ admission control to guarantee the stretch bound, we next evaluate the blocking ratio for each class. We note from Fig. 14 that the blocking ratio increases with load, which is expected. But, the most important point to note is that there is hardly any difference in the blocking ratio between classes. This implies that content is not discriminated against for its class at the call admission part, although, on a relative scale, we note that a higher class (class 0) has a higher blocking ratio. We also note that, at very high load, the blocking ratio almost approaches 0.9. This means that, at a very high load (above 7Mbps), the current stretch bound may not be appropriate. Therefore, stretch bound selection should depend upon the operating range of load. This is because all content can undergo a certain degree of stretch. For example, if load in the system is low, then the content gets more resources and, therefore, not stretched. On the other hand, if the system load is high, the download time will get delayed, i.e., the content will be stretched.

5.2.4 Blocking Ratio on Size

In Fig. 15, we show the dependency of the blocking ratio of each class on content size. We observe that the blocking ratio decreases with the size. This is because the difference in the real and ideal deadline is higher for larger content size. By ideal deadline, we mean the deadline with stretch 1 and the real deadline corresponds to the target stretch bound. This higher difference allows the bigger content to pass the feasibility check with higher probability resulting in a lower blocking rate.

5.2.5 Throughput Comparison

In order to evaluate the throughput, we compared the four scheduling schemes as shown in Fig. 16. As expected, the C/I scheduling provides the maximum throughput since it adapts to instantaneous channel condition. Both

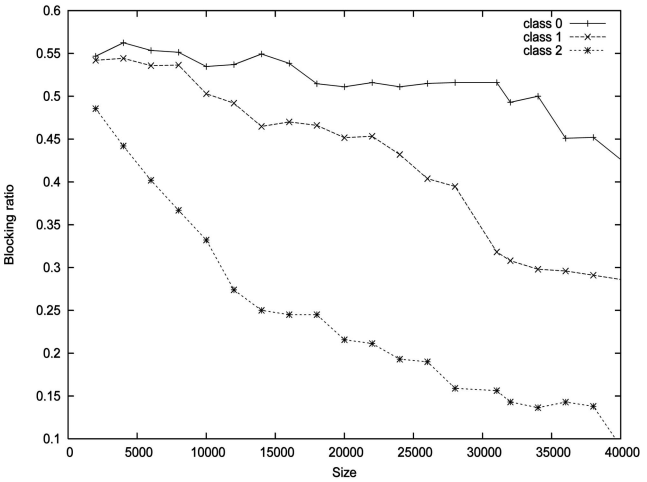

Fig. 15. Blocking ratio versus content size.

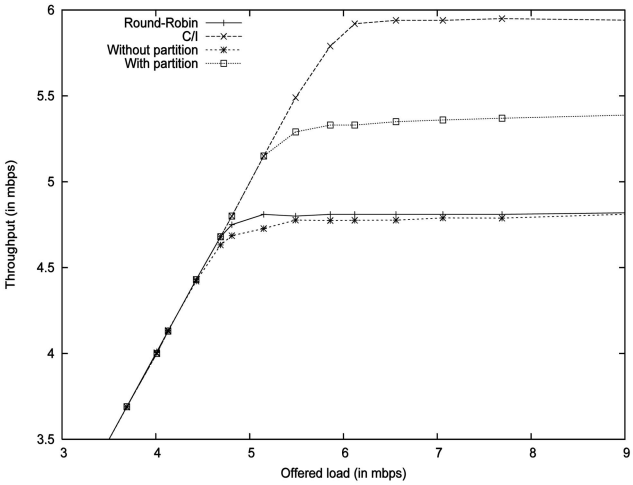

Fig. 16. Throughput comparison for different scheduling schemes.

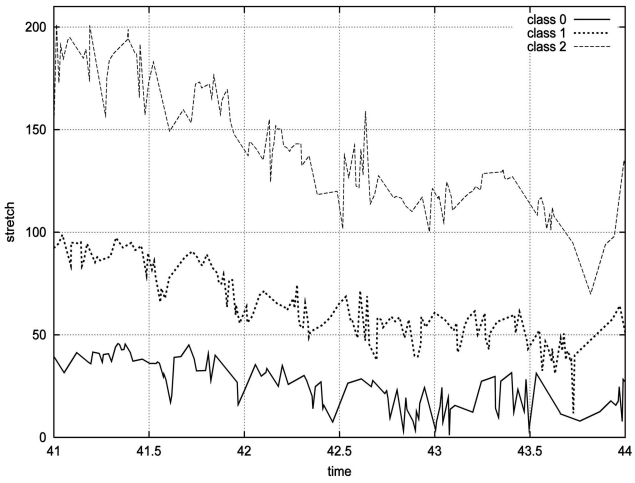

Fig. 18. Stretch for a different class with time.

Round-Robin and stretch scheduling without partitioning provide the worst throughput at high load. The figure also shows that stretch scheduling with block-based partitioning provides a significant improvement in throughput, proving the efficiency of block scheduling.

5.2.6 Comparison with No Admission Control

Here, we try to experiment on the stretch if no admission control is enforced. The results are shown in Fig. 17 for each class under the two cases: with and without admission control. The Y axis in the same figure is shown in log scale and the x axis is the load in Mbps. The figure shows that, without admission control, there is no guarantee on the bound on stretch. However, we observe from the same figure that, without admission control, the stretch for low class saturates at a very high load. This is because of the limited size of the content buffer.

5.3 QoS with Stretch Differentiation

We study the performance of our proposed scheduling framework for four metrics which are relevant for differentiated stretch.

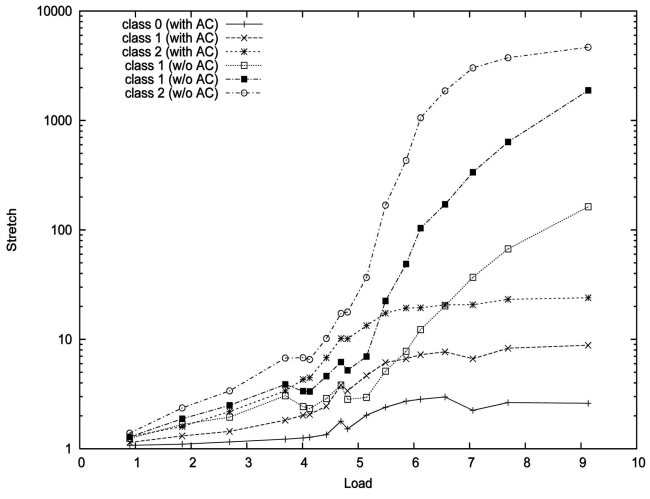

Fig. 17 Stretch comparison with and without admission control.

5.3.1 Transient Stretch Differentiation

Our target is to find out how the stretch of content evolves over time. Due to the bursty nature of data traffic, the loading of the queue will vary abruptly. However, the content stretch scheduling should still provide the necessary differentiation under any instantaneous load condition. Instead of showing for the entire duration, we choose a time window as shown in Fig. 18. We do note that the stretch varies significantly within the time window. However, there is a clear separation between content from different classes. For example, at time = 41.5 s, we note that stretch for class 0 is around 40, for class 1 around 85, and around 175 for class 2. This result thus shows the effectiveness of the scheduling algorithm in maintaining the instantaneous stretch ratio of 1:2:4 among three classes.

One can note that, when the system transient load is low, the stretch ratio is not maintained, as shown in time = 43.5 in Fig. 18. This is because, at that instant, the buffer is underloaded. In the underloaded scenario, the stretch for each class tends to one.

5.3.2 Max/Mean Stretch with Load

In Fig. 19, we show both the mean and maximum values for all the content classes considered. The x-axis shows different load values. In the low load scenario where the queue is empty most of the time, there is no difference in the stretch ratios as expected. When the load increases, the change in the stretch ratios also increases. At overload condition, there is no change in stretch ratio. Next, we observe that the stretch ratios are best maintained between class 1 and 0. On the other hand, for classes 2 and 1, the ratios are lower than target value 4. One of the important observations one can make from Fig. 19 is that there is hardly any difference between max and mean ratios.

5.3.3 Comparison with Pure Packet Scheduling

One of the important reasons that motivates the use of stretch scheduling at content level is the weakness in pure packet scheduling mechanisms. To demonstrate this, we implemented a weighted fair sharing with ratio 4:2:1 for three classes $\{0, 1, 2\}$. These ratios were chosen to make a fair comparison with content scheduling with target stretch

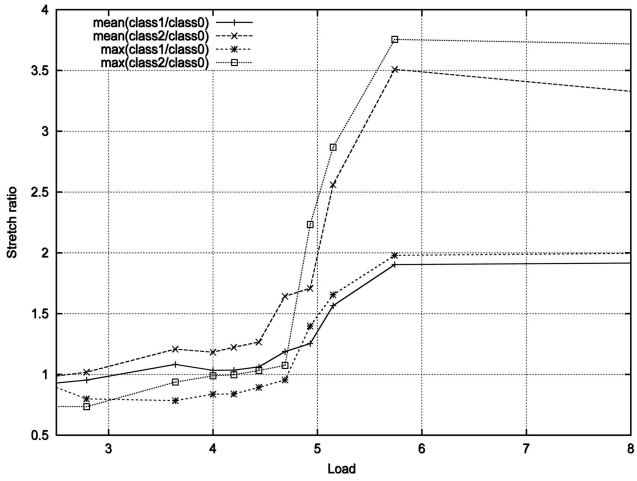

Fig. 19. Stretch ratio versus load for long-term QoS.

ratios. The question is, if we just provide two times the bandwidth to class 0 with respect to class 1, do we get a stretch ratio of 2? To answer this, we show the long-term results under different load scenarios. As Fig. 20 shows, we do get a difference under different loads in the stretch ratios, which also shows that a higher class gets lower stretch. However, we note that the target stretch ratio is not maintained. Further, these ratios are different at different loads with no bound. From a user's point of view, he might get different relative fairness in download time under a different load. These results also demonstrate the need for a content scheduler.

5.3.4 Throughput Comparisons

Next, we show the throughput under different scheduling options in Fig. 21. We observe that, at low load, there is no difference in throughput between scheduling disciplines as expected. At high load, we note that C/I provides the highest throughput. We also note that the throughput of instantaneous nonpartition is the worst since there is no parallelism in content pushing. Even round-robin (RR) performs better than the nonpartition case. This is because, in RR, all the nonempty

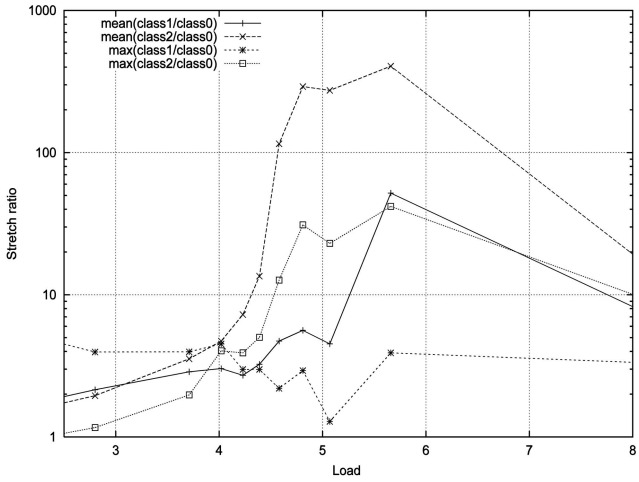

Fig. 20. Stretch ratio versus load with weighted proportional scheduling.

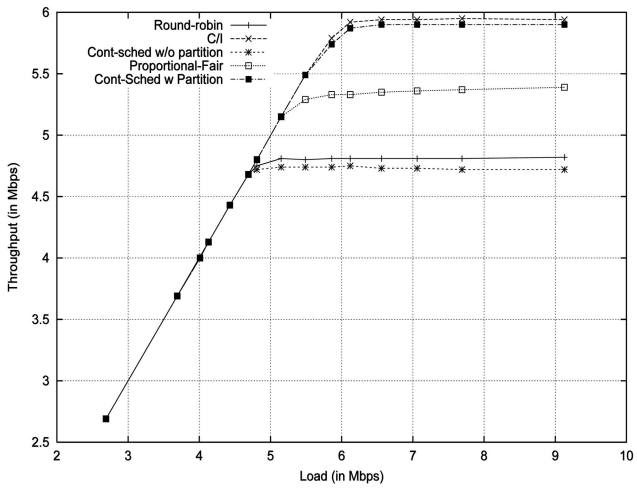

Fig. 21. Throughput comparisons.

queues are served, which results in a higher throughput. We next observe the instantaneous QoS on partition provides better throughput than long-term QoS.

6 Conclusions

The main intent of this research was to explore the service architectures and concepts for non-real-time content delivery for future generation networks. We provided a generalized QoS framework based on the stretch factor that captures the elasticity in the total download time for non-real-time content. We propose a two-stage scheduling architecture along with the scheduling algorithms. We defined the notion of guaranteed and differential QoS and tuned the scheduling algorithms toward providing guaranteed and differential QoS to content based on their QoS class. Furthermore, we presented mechanisms for parallelizing the schedule such that multiple content can be transmitted simultaneously to multiple users. Parallelization of content helps the C/I scheduler work effectively in increasing the throughput. The simulation results indicate that the proposed content scheduling algorithm achieves the target QoS. The results also indicate that, in addition to achieving the target QoS, the proposed scheduling framework provides more throughput than opportunistic proportional fair scheduling policy.

The significance of the proposed service architecture is that it will allow for creation of new wireless services and provide the end user with flexibility to adjust the service quality based on the importance or urgency of the content to be downloaded. As a part of our future work, we intend to explore the pricing issues in such a service architecture based on work done in [18]. A more challenging issue is to address how to do content scheduling in a nonclairvoyant manner where the content size is not known a priori.

References

[1] http://www.3gpp.org, 2006.
[2] http://www.3gpp2.org, 2006.
[3] 3GPP TR25.848 V0.5.0, "Physical Layer Aspects of UTRA High Speed Downlink Packet Access (HSDPA)," ANNEX B, TSGR1#18(01)186, Jan. 2001.

[4] ETSI, UMTS TR101 112, "Selection Procedures for the Choice of Radio Transmission Technologies of the UMTS," 2004.
[5] L. Becchetti, S. Diggavi, S. Leonardi, A. Marchetti, S. Muthukrishnan, and T. Nandagopal, "Parallel Scheduling Problems in Next Generation Wireless Networks," *Proc. ACM Symp. Parallel Algorithms and Architectures,* pp. 238-247, Aug. 2002.
[6] P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, and A. Viterbi, "CDMA/HDR: A Bandwidth-Efficient High-Speed Wireless Data Service for Nomadic Users," *IEEE Comm. Magazine,* pp. 70-77, July 2000.
[7] M. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and Stretch Metrics for Scheduling Continuous Job Streams," *Proc. ACM Symp. Discrete Algorithms,* pp. 270-279, 1998.
[8] S. Borst, "User-Level Performance of Channel-Aware Scheduling Algorithms in Wireless Data Networks," *Proc. IEEE Infocom,* pp. 636-647, 2003.
[9] C. Dovrolis and P. Ramanathan, "A Case for Relative Differentiated Services and the Proportional Differentiation Model," *IEEE Network,* vol. 13, no. 5, pp. 26-34, Sept. 1999.
[10] K. Enoki, "I-Mode: The Mobile Internet Service of the 21st Century," *IEEE Int'l Solid-State Circuits Conf. (ISSCC), Digest of Technical Papers,* pp. 12-15, 2001.
[11] S. Ganguly, N. Tu, M. Chatterjee, and R. Izmailov, "Non-Real Time Content Scheduling in Wireless Data Networks," *Proc. IEEE Int'l Symp. Personal, Indoor, and Mobile Radio Comm. (PIMRC),* vol. 2, pp. 1375-1379, Sept. 2004.
[12] J. Gehrke, S. Muthukrishnan, R. Rajaraman, and A. Shaheen, "Scheduling to Minimize Average Stretch," Technical Report 99-2, DIMACS, Rutgers Univ., 1999.
[13] H. Kushner and P. Whiting, "Asymptotic Properties of Proportional-Fair Sharing Algorithms," *Proc. 40th Ann. Allerton Conf. Comm., Control, and Computing,* 2002.
[14] N. Joshi, S. Kadaba, S. Patel, and G. Sundaram, "Downlink Scheduling in CDMA Data Networks," *Proc. ACM Mobicom,* pp. 179-190, 2000.
[15] A. Konrad, B. Zhao, A. Joseph, and R. Ludwig, "A Markov-Based Channel Model Algorithm for Wireless Networks," *Wireless Networks,* vol. 9, no. 3, pp. 189-199, May 2003.
[16] Y. Liu and E. Knightly, "Opportunistic Fair Scheduling over Multiple Wireless Channels," *Proc. Ann. Joint Conf. IEEE Computer and Comm. Soc. (IEEE INFOCOM),* vol. 2, pp. 1106-1115, 2003.
[17] X. Liu, E.P.K. Chong, and N.B. Shroff, "Opportunistic Transmission Scheduling with Resource-Sharing Constraints in Wireless Networks," *IEEE J. Selected Areas in Comm. (JSAC),* vol. 19, no. 10, pp. 2053-2064, Oct. 2001.
[18] P. Marbach, "Pricing Differentiated Services Networks: Bursty Traffic," *Proc. IEEE Infocom,* pp. 650-658, 2001.
[19] H. Ming, Z. Junshan, and J. Sadowsky, "Traffic Aided Opportunistic Scheduling for Downlink Transmissions: Algorithms and Performance Bounds," *Proc. Ann. Joint Conf. IEEE Computer and Comm. Soc. (IEEE INFOCOM),* vol. 3, pp. 1652-1661, 2004.
[20] S. Parkvall, E. Dahlman, P. Frenger, P. Beming, and M. Persson, "The Evolution of WCDMA towards Higher Speed Downlink Packet Data Access," *Proc. IEEE Vehicular Technology Conf. (VTC),* vol. 3, pp. 2287-2291, 2001.
[21] S. Souissi and S.B. Wicker, "A Diversity Combining DS/CDMA System with Convolutional Encoding and Viterbi Decoding," *IEEE Trans. Vehicular Technology,* vol. 44, no. 2, pp. 304-312, May 1995.
[22] A. Varga, "The OMNeT++ Discrete Event Simulation System," *Proc. European Simulation Multiconf. (ESM 2001),* 2001, http://www.hit.bme.hu/phd/vargaa/omnetpp.htm.
[23] D. Verma, *Content Distribution Networks: An Engineering Approach,* first ed. John Wiley & Sons, 2001.
[24] P. Viswanath, D. Tse, and R. Laroia, "Opportunistic Beamforming Using Dumb Antennas," *IEEE Trans. Information Theory,* vol. 48, pp. 1277-1294, 2002.
[25] H. Zhu, H. Tang, and T. Yang, "Demand-Driven Service Differentiation for Cluster-Based Network Servers," *Proc. IEEE Infocom,* pp. 679-688, 2001.

Samrat Ganguly received the BSc degree in physics from the Indian Institute of Technology, Kharagpur, India, in 1994, the ME degree in computer science from the Indian Institute of Science, Bangalore, India, in 1998, and the PhD degree in computer science from Rutgers University, Piscataway, New Jersey, in 2003. Since 2001, he has been a research staff member at NEC Laboratories America, Princeton, New Jersey. His research interests include distributed algorithm design and performance optimization in wireless, overlay, and content delivery networks.

Mainak Chatterjee received the PhD degree from the Department of Computer Science and Engineering at The University of Texas at Arlington in 2002. Prior to that, he completed the BSc degree in physics (Hons) from the University of Calcutta in 1994 and the ME degree in electrical communication engineering from the Indian Institute of Science, Bangalore, in 1998. He is currently an assistant professor in the School of Electrical Engineering and Computer Science at the University of Central Florida. His research interests include economic issues in wireless networks, applied game theory, resource management and quality-of-service provisioning, ad hoc and sensor networks, CDMA data networking, and link layer protocols. He serves on the executive and technical program committee sof several international conferences.

Rauf Izmailov received the MS degree in mathematics from the Moscow State University, Moscow, Russia, in 1984, and the PhD degree in control science from the Institute of Control Science, Moscow, Russia, in 1988. From 1991 to 1993, he was with AT&T Bell Laboratories, Holmdel, New Jersey. Since 1994, he has been with NEC Labs America, where he is currently a department head of the IP Networks & Distributed Systems Department. His research interests include control problems in communication networks, optimization algorithms, and performance evaluation. He is a member of the IEEE.