A QoS-Aware and Energy-Conserving Transcoding Proxy Using On-demand Data Broadcasting

Jiun-Long Huang
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, ROC
E-mail: jlhuang@cs.nctu.edu.tw

Ming-Syan Chen, Fellow, IEEE
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail: mschen@cc.ee.ntu.edu.tw

Abstract

Most research works in transcoding proxies in mobile computing environments are on the basis of the traditional client-server architecture and do not employ the data broadcast technique. In addition, the issues of QoS provision and energy conservation are also not addressed in the prior studies. In view of this, we design in this paper a QoS-aware and energy-conserving transcoding proxy by utilizing the on-demand broadcasting technique. We first propose a QoS-aware and energy-conserving transcoding proxy architecture, abbreviated as QETP, and model it as a queueing network consisting of three queues. By analyzing the queueing network, three lemmas are derived to estimate the load these queues. We then propose a version decision policy and a service admission control scheme to provide QoS in QETP. The derived lemmas are used to guide the execution of the proposed version decision policy and service admission control scheme to achieve the given QoS requirement. In addition, we also propose a data indexing method to reduce power consumption of clients. To measure the performance of the proposed architecture, three experiments are conducted. Experimental results show that the average access time reduction of the proposed scheme over traditional client-server architecture ranges from 45% to 75%. Experimental results also show that the proposed scheme is more scalable than traditional client-server architecture and is able to effectively control the system load to attain the given QoS requirements. In addition, the proposed scheme is able to greatly reduce average tuning time of clients at the cost of a slight increase (around 5% in our experiments) in average access time.

Key words: Transcoding proxy, QoS, energy-conservation, data broadcast, on-demand broadcast

1 Introduction

In a pervasive computing environment, due to the constraints resulting from power-limited mobile devices and low-bandwidth wireless networks, designing a power conserving mobile information system with high scalability and high bandwidth utilization becomes an important research issue, and hence attracts a significant amount of research attention. In addition, the high diversity in the capabilities of various mobile devices such as display capabilities (e.g., screen size, color depth and supported data formats) and computation power makes the design of mobile information systems more challenging. This diversity also results in an increasing demand on the capability of *context awareness* for mobile information systems.

Content adaptation, which is an important technique to realize context awareness, emerges to remedy the problem resulting from the said diversity by offering different mobile users suitable *versions* of the same object according to the capabilities of the mobile devices, the traffic of the networks and the users' preferences [20]. *Transcoding*, which transforms a data object from one version into another, is recognized as a promising technique to realize content adaptation [20][21][23]. A proxy capable of transcoding (referred to as a transcoding proxy) is placed between a client and an information server to coordinate the mismatch between what the server provides and what the client prefers. Since proxy-based approaches are transparent to the content providers and users, this kind of approach is able to simplify the design of servers and clients, and as a result, attracts much research attention.

In recent years, data broadcast [2][3][29] has been employed as an important technique to design a scalable and power conserving mobile information system. However, most research works in transcoding proxies in mobile computing environments are on the basis of the traditional client-server architecture and do not employ the data broadcast technique. Hence, the transcoding proxies are not scalable and the network bandwidth is not well utilized. In addition, most prior studies do not consider the issue of quality of service (abbreviated as QoS) which is crucial in a mobile computing environment.

In addition, as shown in [26], only a modest improvement (20% $\sim$ 30%) in battery lifetime is expected in the next few years. Hence, energy conservation is raised as a key factor of the design of mobile devices. Since data indexing is recognized as a promising means to reduce power consumption [17], many researchers have studied the design of data indexing algorithms in push-based data broadcasting environments [9][22][28][30]. However, most studies on on-demand data broadcasting focus on the design of scheduling algorithms [1][3], and only a few of them consider the employment of data indexing in on-demand data broadcasting environments [18].

In view of this, we design in this paper a scalable, QoS-aware and energy-conserving transcoding proxy by utilizing the on-demand broadcasting technique. Explicitly, we first propose a QoS-aware and energy-conserving transcoding proxy architecture, abbreviated as QETP, and model it as a queueing network with three queues. By analyzing the queueing network, three lemmas are derived to formulate the average waiting time of these queues. We then devise scheme ODB-QoS-Index to provide QoS in QETP where ODB-QoS-Index stands for "On-demand Data Broadcasting with QoS and data Indexing." Scheme ODB-QoS-Index is an online, iterative and adaptive algorithm comprising

1. a version decision policy to determine the suitable version for each data request according to the users' device profiles and the state of the server,
2. a service admission control scheme to determine whether to grant a service registration or a service handoff according to the state of the server, and
3. a data indexing method to insert data indices into the broadcast program to reduce power consumption of clients.

In each iteration, scheme ODB-QoS-Index estimates the average waiting time of each queue based on the derived results, determines the state of each queue according to the corresponding estimation of average waiting time, and configures the behavior of the version decision policy and the service admission control scheme in accordance with the states of these queues to attain the desired QoS. In addition, scheme ODB-QoS-Index inserts index items into the broadcast program to reduce the clients' power consumption. To measure the performance of QETP, three experiments are conducted. Experimental results show that the average access time reduction of the proposed scheme over traditional client-server architecture ranges from 45% to 75%. Experimental results also show that scheme ODB-QoS-Index is more scalable than traditional client-server architecture, and is able to achieve the system administrators' QoS requirements by the devised version decision policy and the service admission control scheme. In addition, scheme ODB-QoS-Index is able to greatly reduce average tuning time at the cost

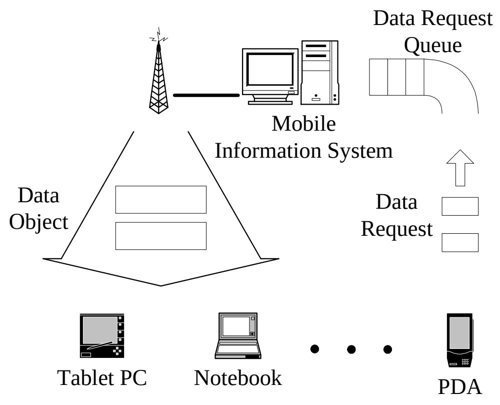


Figure 1: An example on-demand broadcasting system

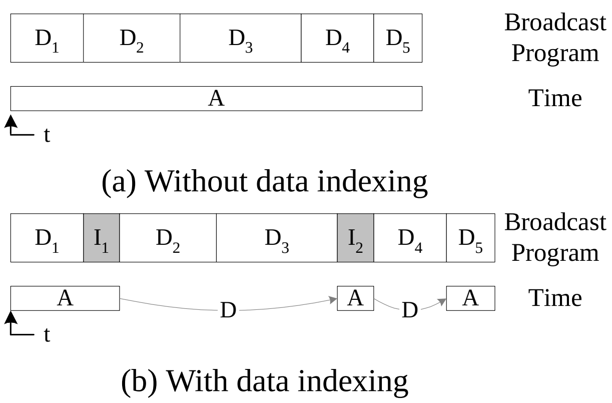


Figure 2: Employment of data indexing

of a slight increase (around 5% in our experiments) in average access time. Access time is defined as the summation of time periods from the moment that mobile clients submit data requests to the moment that mobile clients receive the requested data items. On the other hand, tuning time is defined as the summation of time periods that mobile clients operate in active mode. Access time is widely used to evaluate the efficiency of broadcast systems, while tuning time is used to evaluate power consumption of mobile devices. To the best of our knowledge, there is no prior research on the design of transcoding proxies employing data broadcast. This feature distinguishes this paper from others.

The rest of this paper is organized as follows. The descriptions of related work and the proposed transcoding proxy architecture, QETP, are given in Section 2. An analytical model and a transcoding model are devised in Section 3. Then, Section 4 describes the proposed version decision policy, service admission control scheme and data indexing method. The performance evaluation is shown in Section 5, and finally, Section 6 concludes this paper.

2 Preliminaries

2.1 On-demand Data Broadcasting

Figure 1 shows an example on-demand broadcasting system. In an on-demand data broadcasting system [1][3][4], a server maintains a data request queue and serves these requests according to the employed scheduling algorithm. When requiring one data item, a mobile client sends a data request to the server.

After receiving a data request, the server first checks whether there exists another data request in the data request queue with the same required data object. If yes, the new-coming data request is *merged* into that data request. This phenomenon is called *request merge*. Data requests with the same requested data object can be safely merged since one transmission of the data object in a broadcast channel is able to serve all merged data requests. Therefore, the higher the occurrence probability of request merge is, the more efficient the system is. Otherwise, the new-coming data request is *inserted* into the data request queue.

A scheduling algorithm is used to prioritize all data requests in the data request queue, and the server will serve these data requests according to their priorities. To serve a data request, the system retrieves the required data object from the corresponding data server, and then broadcasts this object to all its clients via a dedicated and shared broadcast channel. As a result, the on-demand broadcast system is more scalable and can obtain higher network utilization than traditional client-server architecture.

2.2 Related Work

2.2.1 Prior Work Related to On-demand Data Broadcasting

Dykeman et al. pointed out in [10] that traditional FCFS scheduling would produce long average access time for an on-demand broadcast system when the access frequencies of all data items were not uniformly distributed. They proposed several scheduling algorithms and concluded that LWF could provide the best performance among the proposed algorithms. Aksoy et al. pointed out in [3] that although being able to produce the shortest average access time, LWF is not efficient when the number of data requests is large. To address this problem, they proposed algorithm RxW which is able to schedule the received data requests efficiently by employing a pruning technique. Experimental results showed that the performance (i.e., average access time) of RxW is close to that of LWF. Unfortunately, the algorithm RxW is designed under the premise that each data item is of the same size. Hence, it is not suitable for variable-sized data items. In [1], Acharya et al. addressed the broadcast scheduling problem in the environments with variable-size data items. They defined a new metric, stretch, as the ratio of the response time of a request to its service time. Based on stretch, they proposed a scheduling algorithm, called LTSF, to minimize the stretch. Wu et al. argued that algorithm LTSF is not optimal

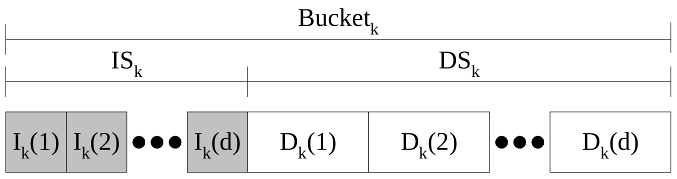


Figure 3: Index structure

in terms of overall stretch [27]. In addition, algorithm LTSF is not scalable in a large-scale environment. Therefore, they proposed a scheduling algorithm to optimize the system performance in terms of stretch. Moreover, the proposed scheduling algorithm is more scalable than LTSF, and hence, is suitable for practical use.

However, most studies on on-demand data broadcasting focus on the design of scheduling algorithms [1][3], and only a few of them consider the employment of data indexing in on-demand data broadcasting environments [18]. Figure 2a and Figure 2b show the examples that a mobile client issues a data request at time t on broadcast programs without and with data indexing, respectively. In Figure 2a and Figure 2b, the time periods marked as ‘A’ and ‘D’ indicate that the time periods that the mobile device is in active and doze mode, respectively. Since the sizes of index items are much smaller than those of data items, employing data indexing is able to greatly reduce the average tuning time at the cost of a slight increase in the average access time.

In [18], Lee et al. proposed a data indexing method in an on-demand data broadcasting environment. As shown in Figure 3, the proposed broadcast program is partitioned into a series of buckets and each bucket contains an index segment and a data segment. The number of the index items in an index segment is equal to the number of data items in the corresponding data segment in the same bucket. In bucket B_k, the i-th index item (i.e., $I_k(i)$) contains (1) the identifier and the version number of the corresponding data item in bucket B_k (i.e., $D_k(i)$), (2) the time offset that $D_k(i)$ will be broadcast and (3) the size of $D_k(i)$. The number of index items within an index segment is called the *degree* of the broadcast program. In [18], the degree of all buckets are the fixed, and the experimental results suggest to set degree of broadcast programs to two for better performance.

2.2.2 Prior Work Related to Transcoding Proxy

Han et al. proposed in [13] an image transcoding proxy which is able to control the data retrieval time to meet users' requirements. The proposed transcoding proxy can adaptively adjust the sizes of the objects transmitted to users by using an aggressive lossy compression method. They also presented an analytical framework for determining whether to transcode and how much to transcode an image, and a process used by the transcoding proxy to adapt its image coding to meet an upper bound on the delay tolerated by the end user.

In [7], Cardellini et al. analyzed how network proxies can work collaboratively in content transcoding and caching. They proposed a distributed algorithm to distribute the computation load caused by transcoding throughout a collaborative proxy system. They also proposed two extended strategies to cache data objects. In [8], Chang et al. explored the aggregate effect when caching multiple versions of the same Web object in the transcoding proxy. They argued that the aggregate profit of caching multiple versions of an object is not simply equal to the sum of the profits of caching individual versions, but rather, depends on the transcoding relationships among them. They devised the notion of a weighted transcoding graph and formulated a generalized profit function. Based on the weighted transcoding graph and the generalized profit function, an innovative cache replacement algorithm for transcoding proxies was proposed, and the proposed cache replacement algorithm was shown to perform well in terms of the delay saving ratios and cache hit ratios.

Hsiao et al. proposed the architecture of versatile transcoding proxy in [14]. Based on the concept of the agent system, the proposed architecture can accept and execute the transcoding preference script provided by the client or the server to transform the corresponding data or protocol according to the user's specification. Fine granularity control is achieved by building a weighted transcoding graph which depicts the transcoding relationship among transcodable versions dynamically. Based on the weighted transcoding graph, the transcoding proxy performs cache replacement according to the content in the caching candidate set, which is generated by the concept of dynamic programming.

In the early study [15] of this paper, we proposed a QoS-aware transcoding proxy architecture to use on-demand broadcast to transmit the requested data objects. However, the issue of energy conservation is not considered. Therefore, for energy conservation, we in this paper extend the prior architecture to

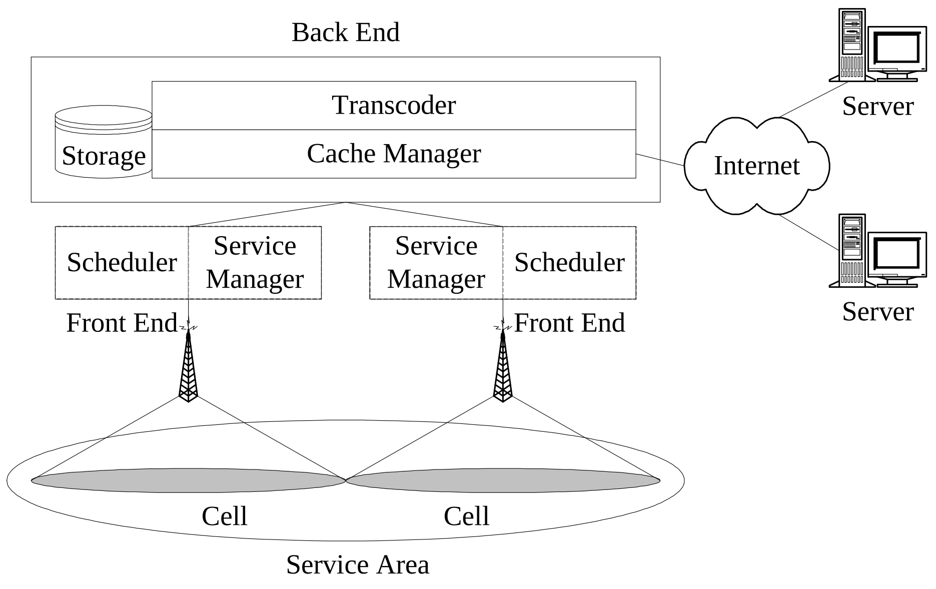


Figure 4: The architecture of QETP

support data indexing techniques. In addition, we also revise the version decision policy and the service admission control scheme proposed in [15] for better performance.

2.3 System Architecture

Figure 4 shows the proposed architecture of QETP. In a cellular environment, the whole service area of a mobile environment is divided into a number of cells. Two dedicated channels, one control channel and one broadcast channel, are provided in each cell. A control channel is used to transmit control messages such as registration messages, data requests, acknowledgements, and so on. On the other hand, a broadcast channel is used by the transcoding proxy to disseminate data objects to its clients. In according to the locations of these components, QETP comprises the following two types of components: front-end and back-end.

A front-end, which comprises a service manager and a scheduler, is allocated to each cell. These two components are described below.

- *Service Manager:* A service manager is in charge of all service-related operations such as service registration, service termination, service admission control and so on. Each service manager owns a profile database storing the users' profiles and the profiles of these users' devices.

- *Scheduler:* A scheduler is a software component which handles the data requests of the corresponding cell. After receiving a data request, the scheduler will first determine a suitable version for this data request according to the user's device profile and the network state. Then, the scheduler will check whether the received data request can be merged to an existing data request in the data request queue. Different from the traditional on-demand broadcasting architecture described in Section 2.1, request merge occurs only when there exists another data request in the data request queue asking for the same version of the same required data object of the received data request. Otherwise, the scheduler will insert the received data request into the data request queue.

  In addition, a scheduling algorithm is employed to determine the service order of the data requests in the data request queue. While serving a data request, the scheduler will send this request to the cache manager and the cache manager will respond with the content of the required data object. The scheduler then broadcasts the required data object via the broadcast channel, and serves the next data request in the data request queue. Moreover, scheduler will broadcast index items through the broadcast channel to reduce the power consumption of mobile clients.

A back-end, which comprises a cache manager and a transcoder, behaves like a traditional transcoding proxy. These two components are described below.

- *Cache Manager:* After receiving a data request from a scheduler, the cache manager is responsible for returning the required version of the required data object to the scheduler. Suppose that the cache manager receives a data request of the j-th version of data object $D(i)$. If the j-th version of D_i is cached, the cache manager will return the cached data object to the scheduler immediately. If the j-th version of D_i is not cached, the cache manager will check whether there exists another version of D_i which can be transcoded into the j-th version of D_i. If yes, the cache manager will ask the transcoder to generate the j-th version of D_i. Otherwise, the cache manager will request the original version of the requested data object from the data server, ask the transcoder to transform the returned data object into the required version, and then transmit the result of transcoding to the scheduler.

- *Transcoder:* A transcoder is in charge of the transformation of data objects among different versions according to the received transformation requests generated by the cache manager.

Since the design of the back-end is similar to the systems proposed in some prior works [7][8][13][25], we focus in this paper on the design of the front-end.

3 Analytical and Transcoding Models

3.1 Analytical Model

In this subsection, we derive the worst case of the average access time[1] of QETP, and use the derived results to propose a version decision policy and a service admission control scheme in Section 4. To facilitate the following discussion, we first make the following assumptions.

1. The employed scheduling scheme of the scheduler is FCFS (standing for first come, first serve).
2. No request merge occurs in the data request queue of the scheduler.
3. One transmission of a data object in the broadcast channel is received by exactly one client.
4. The messages of registration, de-registration and handoff are negligible.

Assumptions 2 and 3 occur when the users' interests are highly diverse, and hence the effect of on-demand broadcast diminishes. We make these two assumptions since we focus on the worst case of the transcoding proxy. Assumption 4 is made since we focus on the situation that the number of data requests is much higher than the number of control messages (i.e., registration, de-registration, handoff and service termination). These assumptions will be relaxed in our simulation model. For better readability, a list of used symbols is shown in Table 1.

We model QETP as a queueing network as shown in Figure 5. Queue 2 is a physical queue which is located in the scheduler. On the contrary, Queue 1 and Queue 3 are logical queues which are only used to model the control and broadcast channels in order to derive the average waiting time of a data request on the control and broadcast channels, respectively. Suppose that the data requests submitted

[1]In this paper we use access time and waiting time exchangeably.

Symbol	Description
P_i	i-th device profile
$D_j(k)$	k-th version of data item D_j
N_{User}	Number of users in the cell
$\lambda_{Ctrl.}$	Aggregate request rate in the cell
$\mu_{Ctrl.}$	Service rate of the control channel
$\mu_{Sche.}$	Service rate of the cache
$\mu_{BCast.}$	Service rate of the broadcast channel
$\rho_{Sche.}$	Standard deviation of the service time of the cache
$B_{Ctrl.}$	Bandwidth of the control channel
B_{BCast}	Bandwidth of the broadcast channel

Table 1: Description of symbols

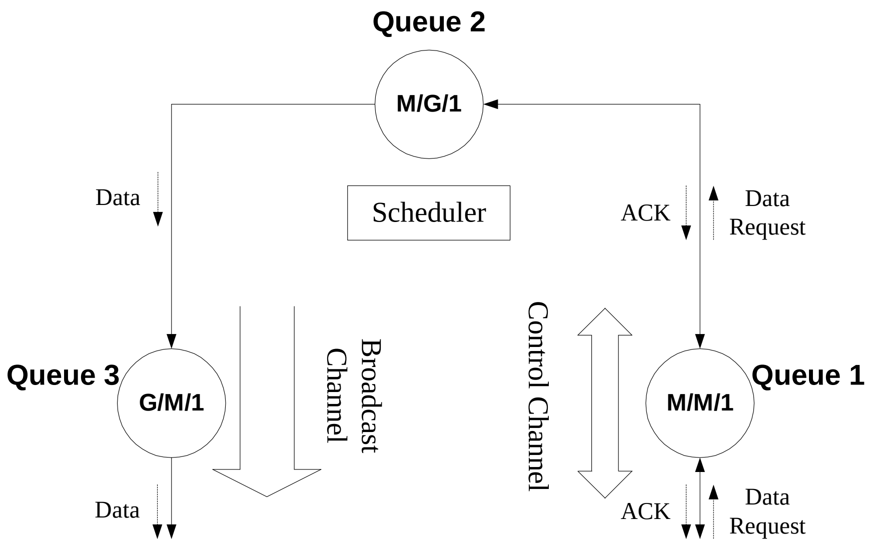


Figure 5: The analytical model of the proposed transcoding proxy

by a mobile user i follow a Poisson process with rate λ_i, and N_{User} is the number of mobile users in the cell. To facilitate the following discussion, we number the mobile users in the cell as user 1, 2, $\cdots$, N_{User}. Due to the characteristic of the Poisson process, the aggregate data requests of all mobile users in the cell follow a Poisson process with rate $\lambda_{Ctrl.} = \sum_{i=1}^{N_{User}} \lambda_i$. Denote the sizes of data requests and request acknowledgements as $s_{Ctrl.}$ and $s_{Ack.}$, respectively. Also let $B_{Ctrl.}$ be the bandwidth of the control channel, and let the waiting time of the control channel for a data request (denoted as $W_{Ctrl.}$) be the time interval between the user sending a data request and the user receiving the acknowledgement. Then, we have the following lemma.

Lemma 1: The average waiting time of the control channel is

$$W_{Ctrl.} = \frac{1}{\frac{B_{Ctrl.}}{s_{Ctrl.}+s_{Ack.}} - \lambda_{Ctrl.}}.$$

Proof: Similar to [19], we assume that the average waiting time to transmit a data request and a request acknowledgement by the control channel is an exponential distribution with mean $\frac{1}{\mu_{Ctrl.}}$. Hence, the control channel can be modeled as an M/M/1 queue. Then, the average service rate of the control channel is

$$\mu_{Ctrl.} = \frac{B_{Ctrl.}}{s_{Ctrl.} + s_{Ack.}}.$$

Omitting the equation manipulation which can be found in [12], the approximated average waiting time for each mobile device from submitting a data request to receiving the corresponding request acknowledgement is

$$W_{Ctrl.} = \frac{1}{\mu_{Ctrl.} - \lambda_{Ctrl.}} = \frac{1}{\frac{B_{Ctrl.}}{s_{Ctrl.}+s_{Ack.}} - \lambda_{Ctrl.}}. \tag{1}$$

Q.E.D.

Let the waiting time of the scheduler for a data request (denoted as $W_{Sche.}$) be, from the scheduler's perspective, the time interval from the arrival of the data request to the time that the requested data object has been obtained. Since the service time of a cache manager is affected by several factors such as cache status of the required data objects, the employed replacement scheme, the characteristics of the input jobs, and so on, the service time of the cache manager cannot be modeled by a particular mathematical distribution. Therefore, we model the average service time of the cache manager as an arbitrary distribution with mean $\frac{1}{\mu_{Sche.}}$ and variance $\sigma^2_{Sche.}$. Let $\rho_{Sche.} = \frac{\lambda_{Ctrl.}}{\mu_{Sche.}}$ be the load of the scheduler. We then have the following lemma.

Lemma 2: The average waiting time of the scheduler is

$$W_{Sche.} = \frac{1}{\mu_{Sche.}} + \frac{\frac{\rho_{Sche.}}{\mu_{Sche.}} + \lambda_{Ctrl.}\sigma^2_{Sche.}}{2(1-\rho_{Sche.})}.$$

Proof: With assumptions 1, 2 and the characteristic of $M/M/1$ queues, the input process seen by the data request queue of the scheduler is also a Poisson process with rate $\lambda_{Ctrl.}$. When receiving a data request, the scheduler determines the most suitable version of the requested data object according to the profile of the mobile device and network status, and then inserts the corresponding job (including data object id and the most suitable version number) into the data request queue. To serve a data request, the scheduler passes the job to the cache manager, and the cache manager will retrieve the specified version of the data object requested by the job and return the retrieved data object to the scheduler. Then, the scheduler disseminates the returned data object to its clients via the broadcast channel.

With assumption 2, the processing of the scheduler can be modeled as an M/G/1 queue. Then, as shown in [12], the expected system size in steady-state is

$$L_{Sche.} = \rho_{Sche.} + \frac{\rho_{Sche.}^2 + \lambda_{Ctrl.}^2 \sigma_{Sche.}^2}{2(1-\rho_{Sche.})}.$$

By Little's formula, the average waiting time of this queue is

$$W_{Sche.} = \frac{L_{Sche.}}{\lambda_{Ctrl.}} = \frac{1}{\mu_{Sche.}} + \frac{\frac{\rho_{Sche.}}{\mu_{Sche.}} + \lambda_{Ctrl.}\sigma_{Sche.}^2}{2(1-\rho_{Sche.})}. \tag{2}$$

Q.E.D.

Let the waiting time of the broadcast channel for a data request be the time interval from the time that the requested data object has been obtained by the scheduler to the time that the user has received it. Then, we have the following lemma.

Lemma 3: The average waiting time of the broadcast channel is

$$W_{BCast} = \frac{1}{\mu_{BCast}(1-r_0)},$$

where r_0 is the root of $z = A^*[\mu_{BCast}(1-z)]$ with value larger than zero and less than one.

Proof: Similar to the proof of Lemma 1, we assume that the average waiting time of the broadcast channel follows an exponential distribution with mean $\frac{1}{\mu_{BCast}}$. Since the broadcast channel is a dedicated downlink channel, similar as [19], we have

$$\frac{1}{\mu_{BCast}} = \frac{\text{Average size of the incoming data objects}}{B_{BCast}}. \tag{3}$$

As shown in Figure 5, the input process of the broadcast channel is the output process of the scheduler. Since the service time of the scheduler (i.e., Queue 2) is an arbitrary distribution, the output process of the scheduler does not follow a particular mathematical distribution. Suppose that the interarrival time of the input process follows an arbitrary distribution with cumulative distribution function $A(t)$. The broadcast channel can be modeled as a G/M/1 queue. Let $A^*(z)$ be the Laplace-Stieltjes transform of $A(t)$. Omitting the mathematical manipulation which can be found in [12], the average waiting time of the broadcast channel (denoted as W_{BCast}) is

$$W_{BCast} = \frac{1}{\mu_{BCast}(1-r_0)}, \tag{4}$$

where r_0 is the root of the following equation with value larger than zero and less than one.

$$z = A^*[\mu_{BCast}(1-z)] \tag{5}$$

Q.E.D.

Finally, the average waiting time of the whole system (denoted as $W_{Sys.}$) is equal to the summation of the average waiting time of the control channel, the scheduler and the broadcast channel. Then, with Lemmas (1), (2) and (3), $W_{Sys.}$ can be formulated as

$$W_{Sys.} \quad = \quad W_{Ctrl.} + W_{Sche.} + W_{BCast} \tag{6}$$

3.2 Transcoding Model

Suppose that the mobile devices are classified into several categories based on their capabilities, and the capabilities of each category are described by one device profile. Let P_i be the i-th device profile. Without loss of generality, we order the device profiles according to their capabilities in ascending order. That is, the capability of P_i is better than that of P_j when $i > j$. We also let $D_i(j)$ be the j-th version of data object D_i. Again, we order all versions of a data object according to their quality in ascending order, which means that the quality of $D_i(j)$ is better than that of $D_i(k)$ when $j > k$. For each data object, we assume that the data size of a version with higher quality is larger than that of another

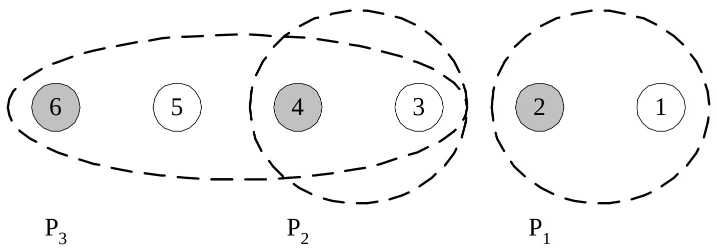


Figure 6: Example device profiles

version with lower quality.

To facilitate the following discussion, the concept of viewable version set is defined below.

Definition 1: A *viewable version set* of a device profile P_i and a data object D_j (denoted as $VVS(i,j)$) is a set of versions of D_j which are able to be displayed by mobile devices with profile P_i.

Then, we have the following example.

Example 1: Consider the example shown in Figure 6. Mobile devices are classified into three categories: notebook, PDA and smart phone, and their capabilities are described in device profiles P_3, P_2 and P_1, respectively. In addition, there are six versions of data object D_j. $VVS(3,j)$, $VVS(2,j)$ and $VVS(1,j)$ are $\{3,4,5,6\}$, $\{3,4\}$ and $\{1,2\}$, respectively. We have $VVS(2,j) \subset VVS(3,j)$ since devices with profile P_3 (e.g., notebooks) are capable of displaying all versions of D_j viewable by devices with profile P_2 (e.g., PDAs). On the other hand, we have $VVS(3,j) \bigcap VVS(1,j) = \phi$ and $VVS(2,j) \bigcap VVS(1,j) = \phi$ since devices with profile P_1 (e.g., smart phone) employ special data formats (e.g., WML and WBMP) that are not supported by devices with profile P_2 and P_3.

Let the function $BEST(i,j) = k$ (respectively, $WORST(i,j) = k$) represent that the best (respectively, worst) viewable version of data object D_j for a mobile device with device profile P_i is version k. In practice, we have $BEST(i,j) \geq BEST(l,j)$ and $WORST(i,j) \geq WORST(l,j)$ when $i > l$. We also have $BEST(i,j) = \max\{VVS(i,j)\}$ and $WORST(i,j) = \min\{VVS(i,j)\}$.

Example 2: Consider the example shown in Figure 6. The best viewable versions of P_3, P_2 and P_1 are $D_j(6)$, $D_j(4)$ and $D_j(2)$, respectively. As a result, we have $BEST(3,j) = 6$, $BEST(2,j) = 4$ and $BEST(1,j) = 2$. In addition, we also have $WORST(3,j) = 3$, $WORST(2,j) = 3$ and $WORST(1,j) = 1$.

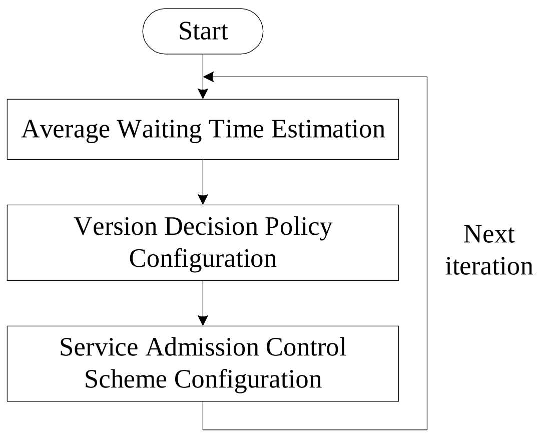

Start
Average Waiting Time Estimation
Version Decision Policy Configuration
Service Admission Control Scheme Configuration
Next iteration

Figure 7: The flowchart of scheme ODB-QoS-Index

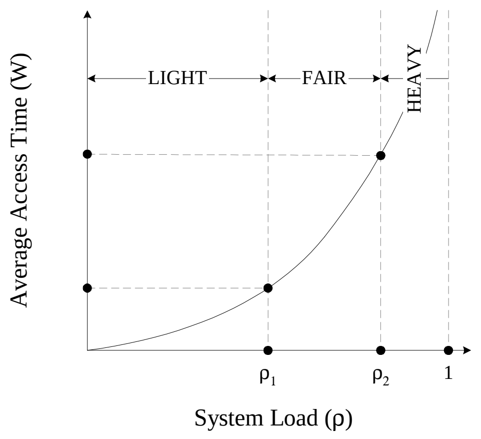


Figure 8: The relationship between load and average access time of a queue

When a user registers the service, the user's mobile device will transmit the identifications of the user and the corresponding device profile to the server. Suppose that the device profile of the mobile device is P_i. Then, when the mobile user requests D_j, the server will return a suitable version of D_j, say the k-th version of D_j where $k \in VVS(i, j)$, according to the result of the underlying version decision policy.

4 Design of Scheme ODB-QoS-Index

An overview of scheme ODB-QoS-Index is given in Section 4.1. The proposed version decision policy and admission control scheme are described in Section 4.2 and Section 4.3, respectively. Finally, the description of the proposed data indexing method is given in Section 4.4.

4.1 Overview

In this paper, we take the average waiting time of the system as the QoS metric. Before executing scheme ODB-QoS-Index, system administrators should specify a QoS requirement by setting two thresholds of average access time, W_1 and W_2 where $W_1 < W_2$. The meanings of these two thresholds are as follows. The users are guaranteed to receive the best viewable versions of the requested data objects when the average waiting time is smaller than W_1. On the other hand, scheme ODB-QoS-Index will try its best to prevent the average waiting time from being larger than W_2.

Scheme ODB-QoS-Index is an online, iterative and adaptive algorithm which comprises a version decision policy, a service admission control scheme and a data indexing method. The flowchart of scheme ODB-QoS-Index is shown in Figure 7. Scheme ODB-QoS-Index is executed periodically, and the following three steps are executed in each iteration. First, in the average waiting time estimation step, scheme ODB-QoS-Index measures the average waiting time of each queue according to the analytical results derived in Section 3. Since only Queue 2 is physical, only the average waiting time of Queue 2 (i.e., $W_{Sche.}$) can be directly observed. In view of this, we propose an approximation algorithm to estimate the average waiting times of Queue 1 and Queue 3 (i.e., $W_{Ctrl.}$ and W_{BCast}). For better readability, the proposed approximation algorithm is described in Appendix A. Then, scheme ODB-QoS-Index measures the load of each queue based on the estimated average waiting time, and determines the current state of each queue according to the load of each queue. Finally, scheme ODB-QoS-Index configures the version decision policy and the service admission control scheme according to the state of each queue. In addition, a data indexing method is employed by the scheduler to insert index items into the broadcast program to reduce power consumption of mobile clients. The details of scheme ODB-QoS-Index are described in the following subsections.

4.2 Version Decision Policy

4.2.1 Overview

Figure 8 shows the relationship between the average waiting time and the load of a queue. It is intuitive that when the load is larger than or equal to one, the system is not stable since the average waiting time does not converge and will approach to infinity. In addition, when the load is smaller than one, the average waiting time increases as the load increases, and the increment will increase drastically when the load approaches one.

With the above observations, the rationale of our scheduling algorithm is *to keep the system loads of the scheduler (i.e., Queue 2 in Figure 5) and the broadcast channel (i.e., Queue 3 in Figure 5) smaller than the predetermined thresholds at the cost of degrading the quality of requested data objects*. As a consequence, when the load of the scheduler or the load of the broadcast channel is high, for each data request, the system will return the version of quality worse than the best viewable version. This strategy has the following two effects:

1. *Decrease the average waiting time of the broadcast channel ($\frac{1}{\mu_{BCast}}$).* Since the data size of a data object with lower quality is usually smaller than that of the same data object with higher quality, transmitting data objects with lower quality is able to reduce the load of the broadcast channel (ρ_{BCast}).

2. *Increase the occurrence probability of request merge.* Consider the device profiles shown in Figure 6, and two data requests of D_j for device profiles P_2 and P_3, respectively. These two data requests will not be merged together when the load of the scheduler or the broadcast channel is light since the system will return the best viewable versions of D_j for P_2 and P_3, respectively. When the load is heavy, the system decides to return the third version of D_j. Hence, these two data requests can be merged together, and the arrival rates of the input processes of the cache and the broadcast channel decrease. As a result, this strategy is able to reduce the load of the cache ($\rho_{Sche.}$) and the broadcast channel (ρ_{BCast}).

The proposed version decision policy consists of three phases: state determination phase, candidate version selection phase and version decision phase. First, in state determination phase the server determines the states of the scheduler and the broadcast channel according to the loads of the scheduler and the broadcast channel. Then, in candidate version selection phase, several versions, called candidate versions, are selected according to the states of the scheduler and the broadcast channel. Finally, the server decides the resultant version from the candidate versions according to the content of the request queue and the objects stored in the cache.

4.2.2 State Determination Phase

Two thresholds, $\rho_1^{Sche.}$ and $\rho_2^{Sche.}$ (respectively, ρ_1^{BCast} and ρ_2^{BCast}), are specified to divide the load of the scheduler (respectively, the broadcast channel) into three states: LIGHT, FAIR and HEAVY. Figure 9 shows the state transition diagram of the scheduler. The state transition scenarios are as follows. When the previous state is LIGHT, the current state will transit to FAIR if $\rho_{Sche.} > (1+\alpha) \times \rho_1^{Sche.}$. Otherwise, the current state will still be LIGHT. When the previous state is FAIR, the current state will transit to

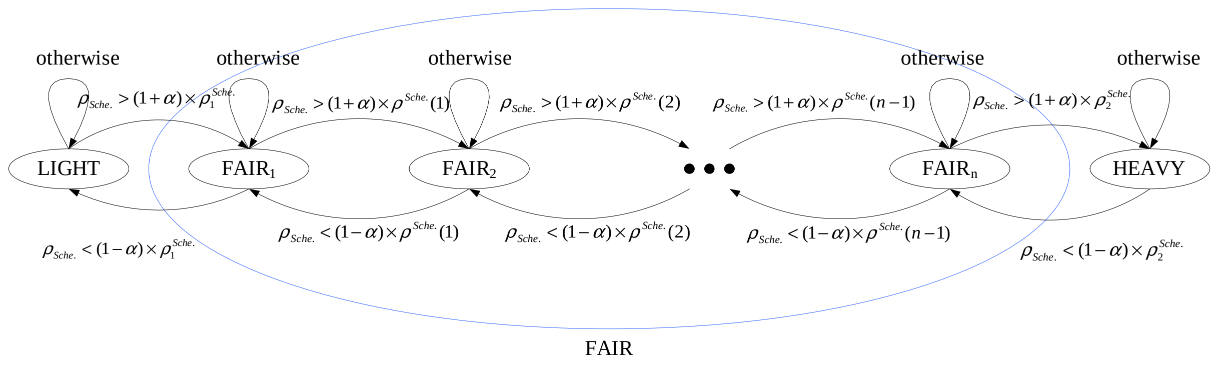

otherwise
otherwise
otherwise
otherwise
otherwise
$\rho_{Sche.} > (1+\alpha)\times\rho_1^{Sche.}$
$\rho_{Sche.} > (1+\alpha)\times\rho^{Sche.}(1)$
$\rho_{Sche.} > (1+\alpha)\times\rho^{Sche.}(2)$
$\rho_{Sche.} > (1+\alpha)\times\rho^{Sche.}(n-1)$
$\rho_{Sche.} > (1+\alpha)\times\rho_2^{Sche.}$
LIGHT
FAIR$_1$
FAIR$_2$
FAIR$_n$
HEAVY
$\rho_{Sche.} < (1-\alpha)\times\rho_1^{Sche.}$
$\rho_{Sche.} < (1-\alpha)\times\rho^{Sche.}(1)$
$\rho_{Sche.} < (1-\alpha)\times\rho^{Sche.}(2)$
$\rho_{Sche.} < (1-\alpha)\times\rho^{Sche.}(n-1)$
$\rho_{Sche.} < (1-\alpha)\times\rho_2^{Sche.}$
FAIR

Figure 9: State transition diagram

LIGHT when $\rho_{Sche.} < (1-\alpha)\times\rho_1^{Sche.}$ and transit to HEAVY when $\rho_{Sche.} > (1+\alpha)\times\rho_2^{Sche.}$. Otherwise, the current state will still be FAIR. When the previous state is HEAVY, the current state will transit to FAIR if $\rho_{Sche.} < (1-\alpha)\times\rho_2^{Sche.}$. Otherwise, the current state will still be HEAVY. The factor α, where $0 < \alpha < 1$, is used to avoid state oscillation. We assume that $(1+\alpha)\times\rho_2^{Sche.} < 1$ without loss of generality. To facilitate fine-grained control, system administrators can divide FAIR state into several sub-states. Suppose that there are n sub-states of FAIR state. The interval $(\rho_1^{Sche.}, \rho_2^{Sche.})$ is then divided into n partitions by $n-1$ thresholds, $\rho^{Sche.}(1), \rho^{Sche.}(2), \cdots, \rho^{Sche.}(n-1)$, where $\rho^{Sche.}(k) = \left(\rho_1^{Sche.} + k\times\frac{(\rho_2^{Sche.}-\rho_1^{Sche.})}{n}\right)$. The transition between these sub-states is similar to that between LIGHT, FAIR and HEAVY states. The state transition diagram and transition scenarios of the broadcast channel are as shown in Figure 9 by substituting ρ_1^{BCast} and ρ_2^{BCast} for $\rho_1^{Sche.}$ and $\rho_2^{Sche.}$, respectively. The determination of the values of $\rho_1^{Sche.}$, $\rho_2^{Sche.}$, ρ_1^{BCast} and ρ_2^{BCast} is described in Appendix B.

We also define the *aggregate state* of the scheduler and the broadcast channel as follows. The aggregate state is LIGHT when the loads of the scheduler and the broadcast channel are both LIGHT. The aggregate state is HEAVY when at least one of the loads the scheduler and broadcast channel is HEAVY. Otherwise, the aggregate state is FAIR. In FAIR state, the current sub-state is determined as the heaviest of the current sub-states (i.e., the heaviest load) of the scheduler and the broadcast channel. For each new-coming data request, the scheduler will decide a suitable version, fill the version information into the data request according to the aggregate state, and insert it into the data request queue. The scheduler will also inform the mobile client of the decided version by replying an acknowledgement message.

4.2.3 Candidate Version Selection Phase

Let $degradation$ and $maxDegradation$ indicate the suggested and maximal degrees of degradation, respectively. The value of $maxDegradation$ is determined by

$$maxDegradation = \max_{\forall P_k, D_j} \{BEST(k,j) - WORST(k,j)\}.$$

In candidate version selection phase, the server will determine a proper value of $degradation$ according to the state of the server, and versions $BEST(k,j), BEST(k,j)-1, \cdots, BEST(k,j)-degradation$ are called candidate versions. The procedure in candidate version select phase is described below.

- *Case I: Aggregate state is LIGHT.*

  The scheduler operates in the traditional on-demand broadcast mode when the aggregate state is LIGHT. Hence, the server guarantees that each client will receive the best viewable versions of the requested data objects. That is, the system will return the $BEST(i,j)$-th version of D_j when a user requests D_j by a mobile device belonging to device profile P_i. Therefore, the value of $degradation$ is set to zero.

- *Case II: Aggregate state is FAIR.*

  In FAIR state, the quality of the received data objects may be degraded. Suppose that FAIR state consists of n sub-states. Then, the value of $degradation$ is set to $\lceil k \times \frac{maxDegradation}{n+1} \rceil$ when the server is in the k-th sub-state of FAIR state.

- *Case III: Aggregate state is HEAVY*

  When the aggregate state is HEAVY, the server will suggest to return the $WORST(i,j)$-th version of D_j when a user requests D_j by a mobile device belonging to device profile P_i. Therefore, the value of $degradation$ is set to $maxDegradation$.

4.2.4 Version Decision Phase

In this phase, the server should pick a proper one from candidate versions (i.e., $BEST(i,j)$, $BEST(i,j)-1, \cdots, BEST(i,j)-degradation$). Suppose that the incoming request is for D_i. The steps of the decision are as follows.

- *Step I:* In this step, the server checks the data requests in the request queue. If in request queue, there is a data request for D_i, say *Req*, with version v, $BEST(i,j) \leq v \leq BEST(i,j) - degradation$, version v is selected since this incoming request can be merged into *Req* without increasing the load of the server. The server will perform step two if there is no such data request in the request queue.

- *Step II:* In this step, the server checks the objects stored in the cache. If there is an object $D_i(v)$, $BEST(i,j) \leq v \leq BEST(i,j) - degradation$, stored in cache, version v is selected so that the server need not neither retrieve D_v from its data server nor perform transcoding. Otherwise, the server will perform step three if there is no such object in the cache.

- *Step III*: Select the version v which is covered by the most profiles among versions $BEST(i,j)$, $BEST(i,j)-1,\cdots,BEST(i,j)-degradation$. Although the server load cannot be reduced by this decision, the probability that successive requests can perform request merge will increase.

4.3 Service Admission Control Scheme

The proposed service admission control scheme consists of two phases: state determination phase and admission control phase. To perform service admission control, the server first determines the state of the control channel in state determination phase, and then determines whether to grant a service registration or a service handoff in admission control phase. The procedures of these two phases are described in the following subsections.

4.3.1 State Determination Phase

The proposed service admission control scheme is employed in each service manager to determine whether to grant a service registration or a service handoff by considering the number of users in service, the network status, and so on. The rate that a service registration is blocked is called *service blocking rate* (abbreviated as SBR), while the rate that a service handoff is forced to terminate is called *service dropping rate* (abbreviated as SDR). The rationale of our service admission control scheme is *to* *keep the system load of the control channel (i.e., Queue 1 in Figure 5) smaller than the predetermined thresholds at the cost of increasing SBR and SDR*. To achieve this, two thresholds, $\rho_1^{Ctrl.}$ and $\rho_2^{Ctrl.}$ where $\rho_1^{Ctrl.} < \rho_2^{Ctrl.} < 1$, are specified to divide the load of the control channel into three states: LIGHT, FAIR and HEAVY. The state transition diagram and transition scenario of the service manager are shown in Figure 9 by substituting $\rho_1^{Ctrl.}$ and $\rho_2^{Ctrl.}$ for $\rho_1^{Sche.}$ and $\rho_2^{Sche.}$, respectively. Similarly, the determination of $\rho_1^{Ctrl.}$ and $\rho_2^{Ctrl.}$ is described in Appendix B.

4.3.2 Admission Control Phase

Although the proposed version decision policy can reduce the loads of the scheduler and the broadcast channel, the effect of the proposed version decision policy is limited since it depends on several factors such as the locality of data requests, the cache size and so on. As a consequence, in addition to the load of the control channel, the service admission control scheme should also take the loads of the scheduler and the broadcast channel into consideration. The procedure in admission control phase is as below.

When the load the control channel is HEAVY, the server will block all service registrations and drop all service handoffs in order to relieve the server load. When the load of the control channel is FAIR or LIGHT, the server will determine the values of two probabilities, $Prob_{Block}$ and $Prob_{Drop}$. Then, a service registration will be blocked with probability $Prob_{Block}$, while a service handoff will be dropped with probability $Prob_{Drop}$. It is the system administrators' responsibility to specify how to determine of the values $Prob_{Block}$ and $Prob_{Drop}$. Let $curState_{Ctrl.}$ be the current state of the control channel, and let $curState_{Agg.}$ be the aggregate state of the scheduler and the broadcast channel. Note that SBR should be sacrificed first since mobile users can tolerate a service registration being blocked rather than a service handoff being forced to terminate (i.e., dropped). Therefore, in each combination of $curState_{Ctrl.}$ and $curState_{Agg.}$, $Prob_{Block}$ should be larger than or equal to $Prob_{Drop}$. An example setting for determining $Prob_{Block}$ and $Prob_{Drop}$ in an environment with three sub-states in FAIR state is given in Table 2.

Consider the case that the server decides to reject a service registration of a service handoff since the server's load cannot afford it. If the owner of the service registration or the service handoff, say user *i*, has the same interest to other users using the service, granting this service registration or the service handoff will not increase the server load since all the user *i*'s requests are expected to be able to

		$curState_{Agg.}$				
		LIGHT	FAIR			HEAVY
			$FAIR_1$	$FAIR_2$	$FAIR_3$	
$curState_{Ctrl.}$	LIGHT	0/0	0/0	0.33/0	0.66/0.15	1/0.3
	FAIR	0/0	0.25/0	0.5/0	0.75/0.3	1/0.6

$Prob_{Block}/Prob_{Drop}$

Table 2: An example setting for determining $Prob_{Block}$ and $Prob_{Drop}$

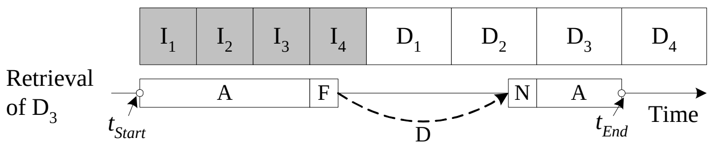
be merged to other users' requests. Hence, to decrease SBR and SDR, the server should grant user *i*'s service registration or service handoff. From the above example, we observe that we can aggressively grant a server registration or a service handoff as long as the owner and other users are of common interest.

To measure the similarity of interest of user *i* and other users using the service, we define similarity factor as the probability that a user's request will be merged to another request. When receiving a data request, the server will check whether the data request is merged into another request and update the user's similarity factor stored in the user's profile. The system administrators have to specify a threshold β, $0 \leq \beta \leq 1$, so that a service registration or a service handoff will be granted (even the server cannot afford it) as long as the value of the owner's similarity factor is larger than or equal to β.

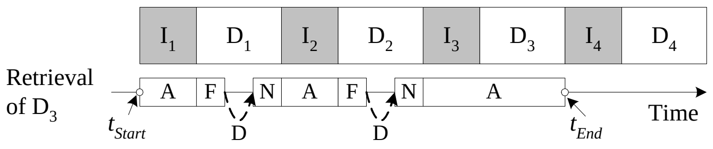
4.4 Data Indexing

As shown in [18], setting degree of broadcast programs to a smaller value will make mobile devices meet index segments more quickly, thus reducing energy consumption. However, it is true only in the cases that turning on and turning off WNIs do not consume energy. As pointed out in [24], in reality turning on and turning off the WNIs consume some time and energy, and the transition times of a WNI from active mode to doze mode and from doze mode to active mode are both on the order of tens milliseconds.

Consider two organizations of index and data items shown in Figure 10. Note that the time periods marked as 'A' and 'D' indicate the time periods that the mobile device is in active and doze mode, respectively, while the time periods marked as 'F' and 'N' indicate that the time periods that the mobile device in turning off and turning on the wireless network interfaces (abbreviated as WNIs). Suppose that a mobile device tunes to the broadcast channel at time t_{Start} and finishes the retrieval of the desired



(a) Example broadcast program with degree four



(b) Example broadcast program with degree one

Figure 10: Example organizations of index and data items

data item at time t_{End}. As observed in Figure 10, when the value of degree of broadcast programs decreases, mobile devices will switch back and forth between active and doze modes (i.e., turn on and turn off WNIs) more frequently, and therefore, may consume more energy. As a result, the value of degree of broadcast programs should be set to a proper value to minimize energy consumption of mobile devices.

In view of this, we adopt an adaptive data indexing method [16] which is able to dynamically adjust the degree of broadcast programs according to system workload. The employed data indexing method consists of two phases, statistics collection phase and degree adjustment phase, and switches back and forth between these two phases periodically. In statistics collection phase, the system collects the arrival time, finish time and other statistical information of each served data request. Then, in the successive degree adjustment phase, the server determines a proper value of degree of broadcast programs according to the collected information. For the interest of space, we omit the description of the determination of the value of degree of broadcast programs. Interested readers can refer to [16] for details.

After determining the current value of degree of broadcast programs, the server then generates the broadcast program accordingly. Since the data items are of different sizes, we use the parameter *budget*, which is defined as the maximal length of the data segments of all buckets, to control the length of each bucket. Initially, the bucket is empty and the scheduler fetches as many data items as possible from the cache under the constraint that the summation of the sizes of the fetched data items is smaller than or equal to *budget*. In addition, the scheduler marks the fetched data items as LOCKED. Then, the scheduler inserts the corresponding index items in front of these data items. Finally, the scheduler broadcasts the index and data items in the bucket sequentially. An index item or a data item is removed from the bucket once it has been broadcast. In addition, the state of a data item which has been broadcast is marked as UNLOCKED. The above procedure repeats until the bucket becomes empty. To employ data indexing, the cache replacement policy should be also modified to consider only data items in UNLOCKED states as the candidates to be replaced.

4.5 Remarks

Currently, the proposed version decision policy and service admission control scheme are designed on the goal of reducing the overall average waiting time and average tuning time. Therefore, if two users submit two data requests (each user submits one request) for the same data object at the same time, their priorities and version numbers will be the same.

It is possible to implement differentiated QoS control in the proposed architecture. For example, we can add a classifier in front of the scheduler to classify the received data requests according to some administrator-specified rules. Hence, the version decision policy is able to assign their version numbers according to their classes. In addition, when processing a service registration or a service handoff, the server first classifies the service according to the user's profile, and then takes action according to the user's class. Consider the case that the server receives two service registrations. Suppose that one is submitted by a VIP user, and the other is submitted by a normal user. The latter will be rejected if the server can accept only one service registration.

5 Performance Evaluation

To evaluate the performance of scheme ODB-QoS-Index, we build an event-driven simulator with SIM [5]. In order to measure the reduction of power consumption of scheme ODB-QoS-Index, we also implement scheme ODB-QoS which only employs the proposed version decision policy and service admission control scheme. Both scheme ODB-QoS-Index and scheme ODB-QoS are executed period-

Parameter	Value
Data object number	4000
Data object sizes	Lognormal dist. (mean 18 KB)
Index item size	1 KB
Data access probabilities	Zipf dist. with parameter 1.1
Cache replacement scheme	AE
Cache capacity	$0.01 \times \sum$object size
Object fetch delay	Exponential dist. with $\mu = 2$
Transcoding rate	30 KB/sec
Client number	1000
Cell residence time	Exp. dist. with $\mu = 40$ minutes
Cell holding time	Exp. dist. with $\mu = 15$ minutes
Cell establishing time	Exp. dist. with $\mu =$ one hour

Table 3: Default system parameters

Profile	Viewable version set
P_1	$\{2,1\}$
P_2	$\{4,3,2,1\}$
P_3	$\{6,5\}$
P_4	$\{8,7,6,5\}$
P_5	$\{10,9,8,7,6,5\}$

Table 4: Device profiles and viewable version sets

ically with period two minutes and the simulation is run for 12 hours. Scheme CS (standing for traditional Client-Server) and scheme ODB (standing for On-Demand Broadcasting) are also implemented for comparison purposes. The average access time and tuning time are employed as the performance metrics of experiments. In addition, the average value of *degradation*, SBR and SDR are taken as the metrics of the cost of scheme ODB-QoS-Index. The average value of *degradation* is used to measure the degree of quality degradation of the received data objects.

5.1 Simulation Model

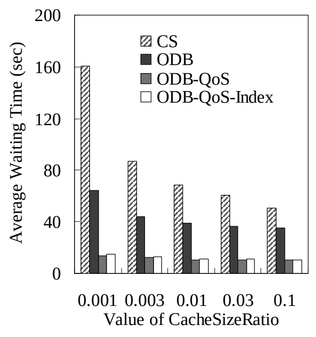
We set the cell topology as a 4×4 cells wrapped-around mesh topology. Scheme AE [8] is employed as the cache replacement policy since it outperforms the other replacement policies for transcoding proxies. Each cell provides one control channel and one download channel with network bandwidth 10 KByte/sec and 100 KByte/sec, respectively. Analogous to [8], we assume that there are 4000 data objects and the sizes follow a lognormal distribution with a mean of 18 KBytes. The sizes of a control message (e.g., data request message and acknowledgement message) and an index item are both set to be 1 KByte. The access probabilities of data objects are assumed to follow a Zipf distribution, which is widely adopted as a model for real Web traces [6]. The parameter of the Zipf distribution is set to be 1.1 with a reference to the analyses of real Web traces [6]. Since small objects are much more frequently accessed than large ones [11], we assume that there is a negative correlation between the object size and its access probability [8]. The default capacity of the cache is set to be “ $0.01 \times \sum$object size” and the fetch delays of data objects follow an exponential distribution with mean two seconds [8]. The values of W_1 and W_2 (i.e., the QoS requirement) are set to be six seconds and 15 seconds, respectively.

In the client model, as in [7] and [8], we assume that the mobile clients are classified into five device profiles, and the distribution of these five device profiles is modeled as a device vector of $\langle 15\%, 20\%, 30\%, 20\%, 15\% \rangle$. Without loss of generality, we also assume that all objects could be transcoded into ten versions, and the sizes of the ten versions (from version one to version ten) are assumed to be 10%, 20%, 30%, $\cdots$ and 100% of the original object sizes [8]. The viewable version set of each device profile is shown in Table 4. By a reference to [8], we assume that a more detailed version can be transcoded into a less detailed one and the transcoding delay is determined as the quotient of the object size to the transcoding rate. The transcoding rate is set to be 30 KBytes/sec [7]. The number of users in the network is set to be 1000. The cell residence time, service holding time and service establishing time for each user are set to be exponential distributions with means of 40 minutes, 15 minutes and one hour, respectively. We also assume that the data requests of each user follow a Possion process with parameter $\frac{1}{\lambda} = 60$ seconds.

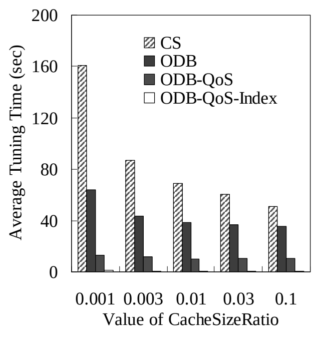
5.2 The Effects of Cache Size

In this experiment, we investigate the effect of varied cache size in average waiting time, average tuning time, SBR, SDR and average value of *degradation*. Figure 11 shows the experimental results with the cache size varied. The cache size is set to be $CacheSizeRatio \times \sum$ object size. The value of *CacheSizeRatio* ranges from 0.001 to 0.1. As shown in Figure 11a, the average waiting time of all schemes decreases as the value of *CacheSizeRatio* increases. This is because the cache with large size is able to effectively reduce the average waiting time by storing data objects with high access probabilities.

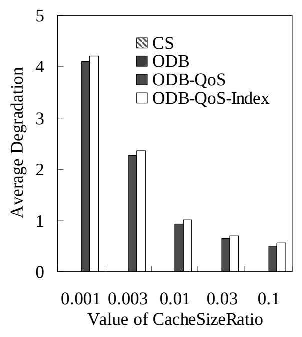
Consider the average waiting time of scheme ODB and scheme CS. The average waiting time reduction of scheme ODB over scheme CS increases from 30% to 60% as the value of *CacheSizeRatio* decreases from 0.1 to 0.001. Since scheme ODB can effectively reduce the number of requests from the cache's perspective by request merge, the system load of scheme ODB is lighter than that of scheme CS. Hence, scheme ODB outperforms scheme CS especially when the cache size is small (i.e., high system load). Although scheme ODB can minimize average waiting time, the performance of scheme



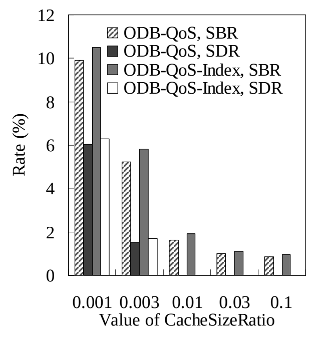
(a) Average Waiting Time



(b) Average Tuning Time



(c) Average Degradation



(d) SBR/SDR

Figure 11: The effect of cache size

ODB does not satisfy system administrators' expectation since the average waiting time is larger than the value of W_2.

To fulfill system administrators' requirement when system load is high, scheme ODB-QoS and scheme ODB-QoS-Index will reduce the quality of the requested data objects and reject some service registrations and service handoffs. Reducing the quality of the requested data objects will increase the probabilities of request merge, and hence reduce the number of data requests from the cache's perspective. In addition, when the system load is still high, scheme ODB-QoS and scheme ODB-QoS-Index will block service registrations to limit the number of users in service. If blocking service registrations still cannot reduce the average waiting time to the administrators' requirement, the service manager will then reject service handoffs. As shown in Figure 11a, the average waiting time of scheme ODB-QoS and scheme ODB-QoS-Index is still smaller than W_2 as the value of $CacheSizeRatio$ decreases. This result shows that scheme ODB-QoS and scheme ODB-QoS-Index are able to control the average waiting time to satisfy the specified QoS requirement. In addition, since scheme ODB-QoS-Index inserts index items into the broadcast program, the average waiting time of scheme ODB-QoS-Index is longer than that of scheme ODB-QoS. Due to the small size of index items, the increment on average waiting time of scheme ODB-QoS-Index over scheme ODB-QoS is quite small (around 5% in this experiment).

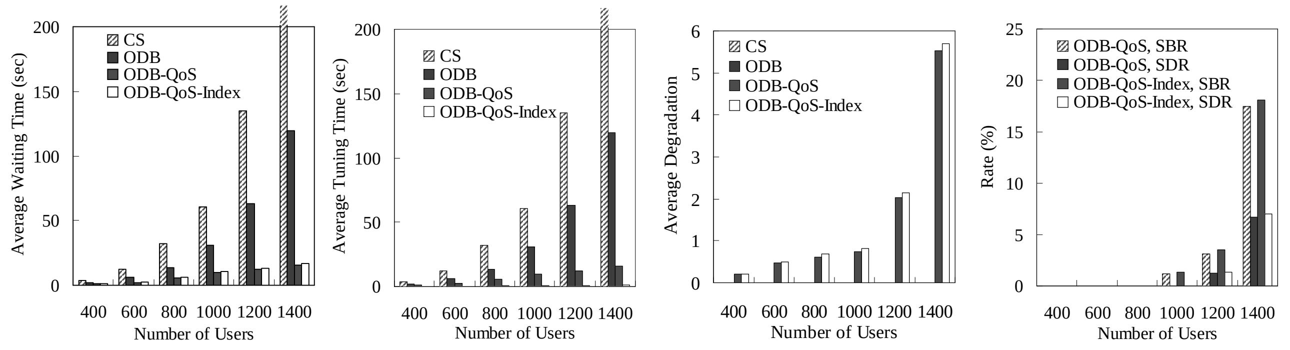
Figure 11b shows the average tuning time of all schemes. Without employing data indexing, the average tuning time and the average waiting time of all schemes except scheme ODB-QoS-Index are the same. In scheme ODB-QoS-Index, when the current bucket does not contain the desired data items, mobile clients can go to doze mode to save power consumption and wake up on the starting point of the next bucket. Therefore, as shown in Figure 11b, scheme ODB-QoS-Index is able to greatly reduce the tuning time (around 93% in this experiment), showing the advantage of data indexing.

Although scheme ODB-QoS and scheme ODB-QoS-Index outperform scheme ODB and scheme CS, scheme ODB-QoS and scheme ODB-QoS-Index produce overhead in SBR, SDR and the degradation on the quality of received data items. Figure 11c and Figure 11d show the degradation on the quality of received data items and the produced SBR and SDR, respectively, of scheme ODB-QoS and scheme ODB-QoS-Index with the value of $CacheSizeRatio$ varied. The SBR and SDR produced by scheme CS and scheme ODB are omitted in this and the following experiments since both schemes always grant service registrations and service handoffs (i.e., both SBR and SDR are always zero).

When the cache size is large enough (i.e., $CacheSizeRatio \geq 0.03$ in this experiment), most hot data items are cached and the average waiting time is under the predetermined threshold. Hence, the average value of $degradation$ is around 0.6 and the quality of the received data items is quite good. In the same condition, SDR is equal to zero and SBR is only a little bit larger than zero. When the cache size becomes small ($CacheSizeRatio = 0.01$ in this experiment), the average value of $degradation$ increases significantly to keep the average waiting time between the predetermined thresholds. When the cache size becomes smaller ($CacheSizeRatio \leq 0.003$ in this experiment), only increasing the value of $degradation$ is not able to effectively relieve the increase of the average waiting time. Hence, the system will block some service registrations to keep the average waiting time under the predetermined threshold. Service registrations is rejected before service handoffs since users can tolerate a service registration to be blocked rather than a service handoff to be dropped. When the value of $CacheSizeRatio$ is very small, some service handoffs are dropped since only blocking service registrations is not able to keep the average waiting time under the threshold. With above mechanisms, scheme ODB-QoS and scheme ODB-QoS-Index are able to keep the average waiting time in the predetermined range.

5.3 The Effects of the Number of Users

Figure 12 shows the experimental results with the number of users varied. The number of users is set from 400 to 1400. From Figure 12a, we observe that when the number of users is small (400 in this experiment), the system load is light and the average waiting times of all schemes are close. When the number of users increases, the average waiting time of scheme CS and scheme ODB also increases.



(a) Average Waiting Time (b) Average Tuning Time (c) Average Degradation (d) SBR/SDR

Figure 12: The effects of the number of users

In addition, the increment of the average waiting time of scheme CS and scheme ODB increases as the number of users increases, especially when the number of users is larger than 1200. Since a large number of users implies high arrival frequencies of data requests, the system load becomes heavy and the average waiting time increases drastically. In this experiment, when the number of users is 1400, the average waiting time of scheme CS does not converge as the time advances since the system load is larger than one. This situation agrees to the observation in Section 4.2. This experimental result also shows that the average waiting time reduction of scheme ODB over scheme CS increases from 47.11% to 74.2% as the number of users increases from 400 to 1400. Scheme ODB is more scalable than scheme CS due to the employment of on-demand data broadcast.

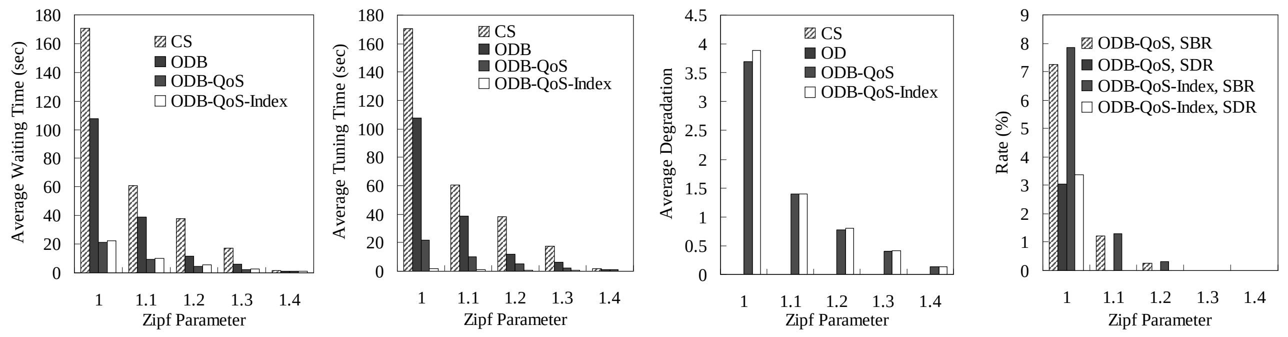
Consider scheme ODB-QoS and scheme ODB-QoS-Index. When the number of users is small (400 in this experiment), scheme ODB, scheme ODB-QoS and scheme ODB-QoS-Index have similar behavior. This can be explained by the reason that when the average waiting time of scheme ODB-QoS is smaller than W_1, scheme ODB-QoS is degenerated to scheme ODB and guarantees that each user will receive the best viewable versions of the requested data objects. In addition, although inserting some index items into the broadcast program, scheme ODB-QoS-Index is still able to perform almost as well as scheme ODB-QoS since the size of index items is much smaller than that of data items. In addition, as shown in Figure 12b, employing data indexing is able to greatly reduce the average tuning time. In this experiment, the tuning time reduction of scheme ODB-QoS-Index over scheme ODB-QoS is around 92%.

As shown in Figure 12c, when the number of users increases to 800, the average value of *degradation* increases in order to keep the average waiting time satisfying the QoS requirement. As shown in Figure 12d, when the number of users increases to 1000, the system blocks some service registrations to satisfy the QoS requirement (i.e., in the interval (W_1, W_2)). Similarly, some service handoffs are dropped when the number of users is larger than 1200. By controlling the quality of received data objects and the number of users in service, scheme ODB-QoS and scheme ODB-QoS-Index are able to keep the average waiting time satisfying the QoS requirement even when the offered system load is heavy.

5.4 The Effects of Skewness of Access Probabilities

Figure 13 shows the experimental results with the skewness of access probabilities varied. The degree of skewness is measured by the value of the Zipf parameter which is set from 1 to 1.4 in this experiment. The larger the Zipf parameter is, the higher the degree of skewness is. As shown in Figure 13a, the average waiting time of all schemes increases as the value of Zipf parameters decreases. It is because that the degree of request locality is high when the access frequencies is highly skewed (i.e., large Zipf parameter). Therefore, with the same cache size, the cache hit ratio is high and is able to effectively reduce the average access time. Moreover, on-demand data broadcasting-based schemes (i.e., scheme ODB, scheme ODB-QoS and scheme ODB-QoS-Index) outperform scheme CS in average waiting time since they take advantage of the locality of data requests by request merge. We also observe that the increment of the average waiting time of scheme CS and scheme ODB increases drastically when the value of Zipf parameter decreases (i.e., one in this experiment). The reason is that the effect of cache and request merge decreases as the degree of skewness decreases. Hence, the system load becomes heavy when the degree of skewness is low, and therefore, the increment of average waiting time increases. This result conforms to the observation in Section 4.2. In this experiment, the average waiting time reduction of scheme ODB over scheme CS ranges from 36.9% to 65%. In addition, as shown in Figure 13b, employing data indexing is able to greatly reduce the average tuning time. In this experiment, the tuning time reduction of scheme ODB-QoS-Index over scheme ODB-QoS is around 90%.

As shown in Figure 13c, the average value of *degradation* is small when access probabilities are



(a) Average Waiting Time (b) Average Tuning Time (c) Average Degradation (d) SBR/SDR

Figure 13: The effects of the Zipf parameters

highly skewed. We also observe from Figure 13d that when the skewness of access frequencies is high (Zipf parameter=1.4 in this experiment), scheme ODB-QoS is degenerated to scheme ODB since the average waiting time of scheme ODB is smaller than W_1. When the access probabilities are not skewed enough, the system cannot fulfill the QoS requirement and will increase the value of *degradation*. When Zipf parameter is around 1.2, some service registrations are blocked (SBR$>$ 0) to satisfy the QoS requirement. Moreover, when Zipf parameter is smaller than 1.1, some service handoffs are also dropped. With the above mechanisms, scheme ODB-QoS and scheme ODB-QoS-Index are able to keep the average waiting time in the predetermined range.

6 Conclusion

We explored in this paper the effect of on-demand broadcasting technique in the design of a QoS-aware and energy-conserving transcoding proxy. We first proposed a QoS-aware and energy-conserving transcoding proxy architecture, QETP, and modeled it as a queueing network. By analyzing the queueing network, several theoretical results were derived to formulate the system average waiting time. We then proposed a version decision policy and a service admission control scheme to provide QoS in QETP. The derived results were used to guide the execution of the proposed version decision policy and service admission control scheme to fulfill the given QoS requirement. In addition, we also proposed a data indexing method to reduce the power consumption of clients. To measure the performance of QETP, several experiments were conducted. Experimental results showed that the proposed scheme is more scalable than traditional client-server systems and can effectively achieve the desired QoS. In addition, the proposed scheme was able to greatly reduce power consumption of clients at the cost of a slight increase in average access time.

References

[1] S. Acharya and S. Muthukrishnan. Scheduling On-demand Broadcasts: New Metrics and Algorithms. In *Proceedings of the 4th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 43–94, October 1998.

[2] M. Agrawal, A. Manjhi, N. Bansal, and S. Seshan. Improving Web Performance in Broadcast-Unicast Networks. In *Proceedings of the IEEE INFOCOM Conference*, March-April 2003.

[3] D. Aksoy and M. J. Franklin. Scheduling for Large-Scale On-Demand Data Broadcasting. In *Proceedings of IEEE INFOCOM Conference*, pages 651–659, March 1998.

[4] D. Aksoy, M. J. Franklin, and S. Zdonik. Data Staging for On-Demand Broadcst. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 571–580, September 2001.

[5] D. Bolier and A. Eliëns. SIM: a C++ library for Discrete Event Simulation. *http://www.cs.vu.nl/~eliens/sim/*, October 1995.

[6] L. Breslau, P. Cao, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of the IEEE INFOCOM Conference*, March 1999.

[7] V. Cardellini, P. S. Yu, and Y.-W. Huang. Collaborative Proxy System for Distributed Web Content Transcoding. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management*, November 2000.

[8] C.-Y. Chang and M.-S. Chen. On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies. *IEEE Transactions on Parallel and Distributed Systems*, 14(6), January 2003.

[9] M.-S. Chen, K.-L. Wu, and P. S. Yu. Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing. *IEEE Transactions on Knowledge and Data Engineering*, 15(1), February 2003.

[10] H.D. Dykeman, M.H. Ammar, and J.W. Wong. Scheduling Algorithms for Videotex Systems Under Broadcast Delivery. In *Proceedings of IEEE ICC Conference*, 1986.

[11] S. Glassman. A Caching Relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27, 1994.

[12] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons,Inc., 3rd edition, 1998.

[13] R. Han, P. Bhagwat, R. Lamaire, T. Mummert, V. Perret, and J. Rubas. Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing. *IEEE Personal Communications*, 5(6), December 1998.

[14] J.-L. Hsiao, H.-P. Hung, and M.-S. Chen. Versatile Transcoding Proxy for Internet Content Adaptation. *to appear in IEEE Transaction on Multimedia*.

[15] J.-L. Huang, M.-S. Chen, and H.-P. Hung. A QoS-Aware Transcoding Proxy Using On-demand Data Broadcasting. In *Proceedings of the IEEE INFOCOM Conference*, March 2004.

[16] J.-L. Huang and W.-C. Peng. An Energy-Conserved On-Demand Data Broadcasting System. In *Proceedings of the 6th International Conference on Mobile Data Management,* May 2005.

[17] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Data on Air: Organization and Access. *IEEE Transactions on Knowledge and Data Engineering*, 9(9):353–372, June 1997.

[18] S. Lee, D. P. Carney, and S. Zdonik. Index Hint for On-demand Broadcasting. In *Proceedings of the 19th IEEE International Conference on Data Engineering*, March 2003.

[19] W.-C. Lee, Q. L. Hu, and D. L. Lee. A Study on Channel Allocation for Data Dissemination in Mobile Computing Environments. *ACM/Kluwer Mobile Networks and Applications*, 4(5):117–129, May 1999.

[20] W. Y. Lum and F. C. M. Lau. A Context-Aware Decision Engine for Content Adaptation. *IEEE Pervasive Computing*, 1(3), July-September 2002.

[21] W. Y. Lum and F. C. M. Lau. On Balancing Between Transcoding Overhead and Spatial Consumption in Content Adaptation. In *Proceedings of the 8th ACM International Conference Mobile Computing and Networking*, September 2002.

[22] W.-C. Peng and M.-S. Chen. Efficient Channel Allocation Tree Generation for Data Broadcasting in a Mobile Computing Environment. *ACM/Kluwer Wireless Networks*, 9(2):117–129, 2003.

[23] C. Poellabauer and K. Schwan. Energy-Aware Media Transcoding in Wireless Systems. In *Proceedings of the 2nd IEEE International Conference on Pervasive Computing and Communications*, March 2004.

[24] T. Simunic, S. Boyd, and P. Glynn. Managing Power Consumption in Networks on Chips. *IEEE Transactions on Very Large Scale Integration Systems*, 12(1):96–107, January 2004.

[25] J. R. Smith, R. Mohan, and C.-S. Li. Content-based Transcoding on Images in the Internet. In *Proceedings of IEEE International Conference on Image Processing*, October 1998.

[26] M. A. Viredaz, L. S. Brakmo, and W. R. Hamburgen. Energy Management on Handheld Devices. *ACM Queue*, 1(7):44–52, October 2003.

[27] Y. Wu and G. Cao. Stretch-Optimal Scheduling for On-Demand Data Broadcasts. In *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*, 2001.

[28] J. Xu, W.-C. Lee, and X. Tang. Exponential Index: A Parameterized Distributed Indexing Scheme for Data on Air. In *Proceedings of the 2nd ACM/USENIX International Conference on Mobile Systems*, June 2004.

[29] J. Xu, X. Tang, and W.-C. Lee. Time-Critical On-Demand Data Broadcast: Algorithms, Analysis, and Performance Evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[30] J. L. Xu, B. Zheng, W.-C. Lee, and D. K. Lee. Energy Efficient Index for Querying Location-Dependent Data in Mobile Broadcast Environments. In *Proceedings of the 19th International Conference on Data Engineering*, March 2003.

Appendix A: Average Waiting Time Estimation

Although the system average waiting time can be formulated by Equation (6), Lemmas 1, 2 and 3, not all components can be directly obtained in practice since Queue 1 and Queue 3 are logical queues. To overcome this problem, we propose an approximation method for each unavailable parameter to estimate the system average waiting time.

Consider the queueing network shown in Figure 5. The input process of Queue 1 cannot be directly observed by the transcoding proxy. However, since the control channel (i.e., Queue 1) is an M/M/1 queue, the output process of Queue 1 is identical to the input process[2] of the corresponding scheduler. Hence, the input process of Queue 1 can be observed by the scheduler, and the average waiting time of the control channel can be obtained by Equation (1). In addition, since the average and variance of the service time of Queue 2 can be observed by the scheduler, the average waiting time of the scheduler can be derived by Equation (2).

To derive the average waiting time of the broadcast channel (i.e., Queue 3), the cumulative distribution function of interarrival time of the input process (i.e., $A(t)$) is required. However, deriving exact $A(t)$ is impractical since $A(t)$ is continuous. Hence, we adopt the following approach to estimate $A(t)$. Consider the m-th execution of scheme ODB-QoS-Index. The average and the variance of the interarrival time of Queue 3 between the $(m-1)$-th and m-th executions (i.e., $\frac{1}{\lambda_{BCast}}$ and σ^2_{BCast}, respectively) can be obtained. We then partition the interarrival time into the following k intervals

$$
\begin{aligned}
I_1 &= \left[\frac{1}{\lambda_{BCast}} - \frac{k-2}{2} \times \sigma_{BCast}, \frac{1}{\lambda_{BCast}} - \frac{k}{2} \times \sigma_{BCast}\right), \\
&\dots \\
I_{\frac{k-1}{2}} &= \left[\frac{1}{\lambda_{BCast}} - \frac{1}{2} \times \sigma_{BCast}, \frac{1}{\lambda_{BCast}} - \frac{3}{2} \times \sigma_{BCast}\right), \\
I_{\frac{k+1}{2}} &= \left[\frac{1}{\lambda_{BCast}} - \frac{1}{2} \times \sigma_{BCast}, \frac{1}{\lambda_{BCast}} + \frac{1}{2} \times \sigma_{BCast}\right), \\
I_{\frac{k+3}{2}} &= \left[\frac{1}{\lambda_{BCast}} + \frac{1}{2} \times \sigma_{BCast}, \frac{1}{\lambda_{BCast}} + \frac{3}{2} \times \sigma_{BCast}\right), \\
&\dots \\
I_k &= \left[\frac{1}{\lambda_{BCast}} + \frac{k-2}{2} \times \sigma_{BCast}, \frac{1}{\lambda_{BCast}} + \frac{k}{2} \times \sigma_{BCast}\right),
\end{aligned}
$$

[2]This phenomenon results from assumption 2 in Section 3.1.

where k is a positive odd number and $k > 1$. Note that although indicating the higher accuracy of the estimation of $A(t)$, a larger k also implies larger memory consumption. We also define an array of $a[1], a[2], \cdots, a[k]$ and reset them to zero in each iteration of scheme ODB-QoS-Index. In the time interval between the m-th and $(m+1)$-th executions, for each data arrival q, the interarrival time of this arrival is counted. If the interarrival time of q lies in interval I_i, the value of $a[i]$ is increased by one. Otherwise, we take the interarrival time of q as an outlier and do not change the values of $a[1], a[2], \cdots, a[k]$. Let $t_p = \frac{1}{\lambda_{BCast}} + (p - \frac{k+1}{2}) \times \sigma_{BCast}$, where $p = 1, 2, \cdots, k$, and $\Delta t = \frac{1}{2} \times \sigma_{BCast}$. We take the discrete distribution with the following probability density function as the approximation of the distribution of interarrival time.

$$f(t) = \begin{cases} \frac{a[i]}{\sum_{j=1}^{k} a[j]} \times \frac{1}{2\Delta t}, & \text{if } t_p - \Delta t \leq t \leq t_p + \Delta t, \text{ where } p \text{ is an integer and } 1 \leq p \leq k; \\ 0, & \text{otherwise;} \end{cases}$$

Let $\frac{1}{\lambda^*_{BCast}}$ be the mean of the approximation of the distribution of interarrival time. We have $\frac{1}{\lambda^*_{BCast}} = \sum_{i=1}^{k} f(t_i) \times t_i$, and then, Equation (5) can be rewritten as

$$z = \sum_{i=1}^{k} a[i] \times e^{-\frac{1}{\lambda^*_{BCast}} t_i (1-z)}. \tag{7}$$

The value of r_0 can be estimated by applying successive substitution in Equation (7). In addition, since the average size of incoming data objects can be observed, the μ_{BCast} can be obtained by Equation (3). Finally, the approximated W_{BCast} can be calculated by Equation (4).

Appendix B: Configuration of the Version Decision Policy and the Service Admission Control Scheme

Here, we develop an adjusting algorithm to configure the version decision policy and the service admission control scheme based on the system state by automatically adjusting the parameters used in Section 4.2 and 4.3. In the proposed adjusting algorithm, three positive factors, γ_1, γ_2 and γ_3 where $\gamma_1 + \gamma_2 + \gamma_3 = 1$, are defined to determine the values of $\rho_1^{Ctrl.}$, $\rho_2^{Ctrl.}$, $\rho_1^{Sche.}$, $\rho_2^{Sche.}$, ρ_1^{BCast} and ρ_2^{BCast}.

The values of $\rho_1^{Ctrl.}$ and $\rho_2^{Ctrl.}$ are first determined so that the average waiting time of the control channel is equal to $W_{Ctrl.} = \gamma_1 \times W_1$ and $W_{Ctrl.} = \gamma_1 \times W_2$, respectively. By substituting $\gamma_1 \times W_1$ for $W_{Ctrl.}$ in Equation (1), we can solve the above equation since $\lambda_{Ctrl.}$ is the only unknown variable in the above equation. Assume that the solution of $\lambda_{Ctrl.}$ is $\lambda_1^{Ctrl.}$. Then we can obtain the value of $\rho_1^{Ctrl.}$ since $\rho_1^{Ctrl.} = \frac{\lambda_1^{Ctrl.}}{\mu_{Ctrl.}}$. With similar approach, $\lambda_2^{Ctrl.}$ and $\rho_2^{Ctrl.}$ can also be obtained.

The values of $\rho_1^{Sche.}$ and $\rho_2^{Sche.}$ are then determined so that the average waiting time of the cache is equal to $\gamma_2 \times W_1$ and $\gamma_2 \times W_2$, respectively. Due to assumption 2 and the characteristic of Queue 1 (i.e., an M/M/1 queue), the input process of Queue 2 is the same as the input process of Queue 1 (i.e., $\lambda_{Sche.} = \lambda_{Ctrl.}$). We rewrite Equation (2) by substituting $\frac{\lambda_{Ctrl.}}{\mu_{Sche.}}$ and $\lambda_1^{Ctrl.}$ for $\rho_{Sche.}$ and $\lambda_{Ctrl,}$, respectively. Then the only unknown variable (i.e., $\mu_{Sche.}$) in the rewritten equation can be solved. Suppose that the solution is $\mu_1^{Sche.}$, and we have $\rho_1^{Sche.} = \frac{\lambda_1^{Ctrl.}}{\mu_1^{Sche.}}$. Analogously, the value of $\rho_2^{Sche.}$ can be obtained by similar approach.

Finally, the values of ρ_1^{BCast} and ρ_2^{BCast} are determined so that the average waiting time of the cache is equal to $\gamma_3 \times W_1$ and $\gamma_3 \times W_2$, respectively. To determine ρ_1^{BCast}, we first rewrite Equation (4) by replacing $\gamma_3 \times W_1$ with W_{BCast}, and the only unknown variable r_0 can be solved. Since γ_3 and W_1 are larger than zero, r_0 is smaller than one. If $r_0 \leq -1$, it indicates that the requirement is infeasible since the required average waiting time of the broadcast channel is under the lower bound. Then, the value of ρ_1^{BCast} is set to 0. Otherwise, when $0 < r_0 < 1$, we rewrite Equation (7) by replacing the solved r_0, $t_i + \delta$ and $\lambda_{BCast}^* + \delta$ with z, t_i and λ_{BCast}^*, respectively. Then, the only unknown variable δ can be solved. Finally, we have $\rho_1^{BCast} = \frac{\lambda_{BCast}^* + \delta}{\mu_{BCast}}$. The value of ρ_2^{BCast} can also be derived by similar approach.

The values of γ_1, γ_2 and γ_3 are determined adaptively and automatically. When the system starts up, γ_1, γ_2 and γ_3 are initialized to $\frac{1}{3}$. In each execution, they are determined by $\gamma_1 = \frac{W_{Ctrl.}}{W_{Sys.}}$, $\gamma_2 = \frac{W_{Cache.}}{W_{Sys.}}$ and $\gamma_3 = \frac{W_{BCast.}}{W_{Sys.}} = 1 - \gamma_1 - \gamma_2$. Note that in scheme ODB-QoS-Index, only QoS requirement (i.e., W_1 and W_2) is required to be specified by system administrators.

Appendix C: Signalling Procedures

Before using the transcoding proxy, a mobile user should register the service in advance by sending a registration message via a control channel. After the transcoding proxy receives the registration message, a service admission control scheme is activated to determine whether to grant the service registration. If yes, the mobile device will send the device profile to the proxy, and the proxy will record the user profile and device profile in its profile database. Otherwise, the service registration is blocked. The rate that a service registration is blocked is called *service blocking rate* (abbreviated as SBR).

After the service registration is granted, the mobile user can issue data requests to the corresponding transcoding proxy by the control channel. When receiving a data request, the transcoding proxy first determines the suitable version of the requested data object by a version decision policy, and returns an acknowledgement message containing the decided version information via the control channel to the mobile user. Then, the transcoding proxy will return the decided version of the required data object via the corresponding broadcast channel as soon as possible. After receiving the acknowledgement message, the mobile device will tune to the broadcast channel to wait for the appearance of the decided version of the requested data object. When the mobile user decides not to use the transcoding proxy service, the mobile device will send a de-registration message to terminate the service.

Since a mobile user is able to freely move around these cells, a service handoff will occur. A service admission control scheme is executed to determine whether the service handoff is granted. If yes, the mobile user can use the service as usual. If not, the system will force the mobile user to terminate the service (the service is said to be dropped). Since a service admission control scheme is employed, a service handoff may be rejected. The rate that a service handoff is forced to terminate is called *service dropping rate* (abbreviated as SDR).

Appendix D: Algorithmic Forms of the Proposed Algorithms

Version Decision Policy

As a consequence, the algorithmic form of the version decision policy is as below.

```
Procedure VersionDecision($P_i$, $D_j$)
Input: A user requests $D_j$ by a mobile device belonging to device profile $P_i$.
Output: A version of $D_j$.
 1: Let $curState_{Agg.}$ be the current aggregate state of the scheduler and the broadcast channel.
 2: $maxDegradation \leftarrow \max_{\forall P_k} \{BEST(k,j) - WORST(k,j)\}$
 3: if ($curState_{Agg.}$=LIGHT) then
 4:     $degradation \leftarrow BEST(i, j)$ /* The system will return the best viewable version to the user */
 5: else if ($curState_{Agg.}$=HEAVY) then
 6:     $degradation \leftarrow WORST(i, j)$ /* The system will return the worst viewable version to the user */
 7: else /* $curState_{Agg.}$=FAIR */
 8:     Determine the sub-state. /* Suppose the aggregate state is in the $k$-th sub-state of FAIR state */
 9:     $degradation \leftarrow \lceil k \times \frac{maxDegradation}{n+1} \rceil$
10: if (rule one can be applied) then
11:     Perform step one to determine the version $v$.
12: else if (rule two can be applied) then
13:     Perform step two to determine the version $v$.
14: else
15:     Perform step three to determine the version $v$.
16: return $v$
```

Service Admission Control Scheme

The algorithmic form of the proposed service admission control scheme is as below.

Procedure ServiceAdmission

Input: A service registration or a service handoff.

Output: Decision of the incoming service registration or service handoff.

```
 1: Let $curState_{Ctrl.}$ be the current state of the control channel.
 2: Let $curState_{Agg.}$ be the current aggregate state of the scheduler and the broadcast channel.
 3: Let $similarity$ be the similarity factor of the owner of the service registration or the service handoff.
 4: if ($curState_{Ctrl.}$=HEAVY) then
 5:     $decision \leftarrow REJECT$
 6: else /* $curState_{Ctrl.}$=FAIR or $curState_{Ctrl.}$=LIGHT */
 7:     Determine the values of $Prob_{Block}$ and $Prob_{Drop}$ according to system administrators' settings.
 8:     if (service registration) then
 9:         Set $decision$ to REJECT with probability $Prob_{Block}$ and to GRANT with probability $(1 - Prob_{Block})$.
10:     else /* service handoff */
11:         Set $decision$ to REJECT with probability $Prob_{Drop}$ and to GRANT with probability $(1 - Prob_{Drop})$.
12: if ($decision$==REJECT) then
13:     if ($similarity \geq \beta$) then
14:         return GRANT
15:     else
16:         return REJECT
17: else /* $decision$==GRANT */
18:     return $decision$
```

Broadcast Program Generation Algorithm

The algorithmic form of the proposed broadcast program generation algorithm is as below.

Algorithm ProgramGeneration

```
 1: while (true) do
 2:     $bucket$ ←BucketGeneration();
 3:     while ($bucket$ is not empty) do
 4:         $item$ ←the head of $bucket$;
```

```
5:         Remove the head of $bucket$;
6:         Broadcast $item$
7:         **if** ($item$ is a data item) **then**
8:             Mark $item$ as UNLOCKED;
```

Function BucketGeneration()

```
1:  $available \leftarrow budget$;
2:  $budket \leftarrow empty$;
3:  **while** (true) **do**
4:      Fetch a data item (denoted as $item$) from the cache;
5:      Mark $item$ as LOCKED;
6:      **if** ($available \geq$the summation of the sizes of $item$ and the corresponding index item) **then**
7:          Append $item$ into $bucket$;
8:          $available \leftarrow available -$the size of $item$-the size of the corresponding index item;
9:      **else**
10:         **if** ($bucket$ is $empty$) **then**
11:             Append $item$ into $bucket$;
12:         **break**;
13: Insert the corresponding index items of the data items in $bucket$ into the head of $bucket$;
14: **return** $bucket$;
```