

# Pervasive Web Content Delivery with Efficient Data Reuse

Chi-Hung Chi, Yang Cao  
*School of Computing*  
*National University of Singapore*  
*Email: chich@comp.nus.edu.sg*

## Abstract

The heterogeneity and dynamics of client's requirement and preference make the current "one content presentation to many global clients" model for information provisioning difficult to deploy cost-efficiently in pervasive and mobile computing environment. Server side solution of maintaining multiple distinct versions of the content is quite limited and expensive; client side solution of transforming the original content at the browser consumes unnecessary bandwidth and might require too much compute power from low-end pervasive devices. Although real-time proxy-based transcoding solution is generally agreed to be the right direction to address this problem, two issues yet to be solved are the high real-time overhead to transcode streaming web data and the inability to reuse one transcoded version of an object to build another transcoded version. In this paper, we propose a new proxy system architecture together with its supported new multimedia data model to achieve scalable pervasive Internet access with efficient data reuse. To further illustrate the importance and practicability of our proposal, we use JPEG2000 as an example and show how it can be implemented in the proxy cache to achieve data reusability and real-time transcoding on streaming web data with minimum overhead.

## 1. Introduction

The widespread use of Internet leads to exploitation of the potential benefits of multimedia, and in turn the proliferation of multimedia on the WWW. Moreover, with hardware such as digital cameras and scanners increasingly inexpensive, people are commonly creating web pages rich in multimedia data. Today, the web is immersed in media of all kinds, including images, audio and video clips. By some estimates [15], about 77% of the data bytes accessed on the web are from multimedia objects. Of these, 67% of the data bytes accessed across the web are transferred for images. However, despite its increasing popularity, migrating multimedia data onto the WWW is not as a simple task as it may seem to be.

From the content provider's perspective, while coarse presentations may disappoint high-end clients, transmitting presentations of excessively high quality to low-end clients wastes not only network bandwidth but also clients' time and money. In either case, the content provider risks to lose his potential customers. As a single multimedia presentation is not able to accommodate the wide spectrum of client capabilities, content providers often find themselves in a dilemma.

From the client's standpoint, the desired presentation of a multimedia object is determined by a number of factors, including processing, storage and display capabilities, network bandwidth, and its personal preference. For instance, client devices using high-speed networks and high-quality displays may wish to

view the images at the highest quality. In contrast, when the network bandwidth is insufficient, clients tend to trade off quality for shorter latency. Even worse, the "best-fit" presentation of a multimedia object to the same client might change with time and environment. Thus, to find one multimedia presentation to fit all kinds of web clients is not easy, if possible. This problem of providing the best-fit presentation to web clients is getting more serious with pervasive Internet access and mobile computing, where variations to factors that determine the best-fit presentation are getting larger.

Some sites address this issue by providing multiple versions of the same multimedia object, each of which fulfils different client's requirement. However, this approach has a number of drawbacks. Due to resource constraints, a server can only provide a limited number of variations. Thus it leaves out a wide middle ground of presentations, as it does not accommodate clients whose configurations suggest a data presentation somewhere in between those available at the server. Another problem is with the granularity of various versions. Fine granularity causes the maintenance of such web sites difficult while large granularity may fail to provide satisfactory service to the clients. From the servers' perspective, maintenance of multiple versions is also troublesome and costly. As clients' profiles keep changing over time, the set of popular versions needs to be updated regularly. Since every update involves a scan of the entire collection, it is exceedingly taxing on the system resource.

In view of the above, it is highly desirable to have a solution that can dynamically adapt the content to client's individual requirement, without incurring significant overhead in time. Such solution not only makes the best use of the network bandwidth, but also relieves the content providers from routine web-site maintenance. This results in the research direction of *real-time transcoding in proxy gateway*. The term transcoding refers to the process by which a multimedia object is converted from one form to another, frequently trading off object fidelity for size and retrieval latency. Transcoding is widely employed to achieve on-the-fly adaptation on multimedia objects.

Transcoding can be deployed in one of the following control points along network path: client, server or intermediate proxy server. Apparently, deploying transcoding at the client is the worst solution, as by this point the maximum amount of bandwidth and loading time will have already been committed. Server is a better option because the utilization of network bandwidth is under control. However, this solution does not address the issue of how data reuse among various transcoded versions of the same object can be achieved in the network proxy caches. Furthermore, proxy gateways (in particular forward proxies) are supposed to understand client's requirement and preference much more than web servers do.

In comparison, proxy server provides an excellent control point in the network path for a number of reasons. Firstly, similar to that in server, deploying transcoding in proxy server helps effectively utilize network bandwidth. More importantly, with migrating transcoding into proxy, all the complexities are pushed away from both client and server, thereby freeing the two ends from resource-intensive computation. Thirdly, deployed at the proxy, caching mechanism can help further reduce bandwidth consumption and transcoding frequency through data reuse. Lastly, the installation and maintenance cost is minimized, as there exists only a single copy of the transcoding application in the entire system. Thus deploying transcoding in the proxy is considered to be a very cost-effective solution.

Several systems have been developed for adapting multimedia objects to client devices through a transcoding system deployed at the proxy. A typical workflow of these systems is as follows: the proxy retrieves the target multimedia object on client's behalf, performs appropriate transcoding (e.g. GIF to JPEG conversion and JPEG compression) on the object, and then transmits the transcoded object to the client. In particular, their transcoding processes cannot commence until the object has been downloaded in its

entirety. This restriction handicaps the potential benefits of these systems in terms of both network bandwidth and user load time for the following reasons:

- As such transcoding only works on full objects, the entire object has to be transmitted from a server to the proxy before the transcoding takes place. Thus while the bandwidth consumption over the client-proxy link is optimized through transcoding, the bandwidth consumption over the proxy-server link is not reduced at all.
- Worse still, such transcoding breaks the pipeline of data blocks transferred across the network. Current HTTP protocol allows object data to be transmitted block by block in streaming fashion from a server to a client. In previous transcoding systems, such streaming is interrupted as the proxy does not pipeline the data transferred from the server to the client but instead waits till the object is downloaded completely. Such waiting delay greatly degrades the system performance.
- The transcoding process itself results in latency varying from hundreds of microseconds to a couple of seconds. Thus the overall latency experienced by the user can be considerably large, or even intolerable over slow network link.
- Furthermore, current transcoding techniques do not allow transcoded objects to be reused. In current transcoding systems, if a client requests for a low quality version of an object, the system transcodes the original object accordingly. If at a later time another client requests for another quality version of the same object, the system has to perform the transcoding from scratch, rather than making use of the previously transcoded object. Apparently this limits the benefits of transcoding in terms of the bandwidth usage and latency.

To address the above problems for real-time transcoding in the network, we make observation to the two emerging trends in multimedia data format and HTTP protocol development. Among the most revolutionary development in multimedia data formats is the advent of progressive data formats, which organize data as a succession of layers, with each additional layer offering one quality increment over its previous layer. One big benefit of progressive data formats is its ability to reconstruct data at various resolutions without the necessity of downloading the complete object. This implies data reuse among various transcoded versions of an object. Furthermore, the overhead of data presentation construction is

negligible, as the transcoding process only involves either reading a data range from an object or cascading two data ranges of an object together. In addition, as recent HTTP/1.1 protocol incorporates the capability that allows a client to request portions of an object via Range requests, transcoding of multimedia objects can be migrated onto the Internet infrastructure without breaking the streaming feature of network data transmission.

In this paper, we propose a new proxy system architecture together with its supported new multimedia data model to achieve scalable pervasive Internet access with efficient data reuse. To further illustrate the importance and practicability of our proposal, we use JPEG2000 as an example and show how it can be implemented in the proxy cache to achieve data reusability and real-time transcoding on streaming web data with minimum overhead.

The rest of this paper is organized as follows. Section 2 reviews related work in progressive multimedia data formats, content adaptation mechanisms and systems. In Section 3, we propose a data model that leads to optimal transcoding on multimedia objects with maximum reuse of data. With the support from our proposed data model, we present a transcoding system architecture that fulfils the following requirements: content adaptation, minimum bandwidth requirement and real-time delivery. In Section 4, we highlight the scalability feature of JPEG2000 – the emerging digital imaging standard, and illustrate how to handle partial object retrieval under HTTP/1.1 protocol. In Section 5, we conclude the work we have done and present some ideas for future work.

## 2. Related Work

In this section, we outline related work on progressive image formats. We also discuss the techniques that are widely employed to address concerns related to the Internet transmission such as network bandwidth, client latency and content adaptation. The recent trend is towards transcoding, as it is the primary technique for achieving content adaptation. We also review a number of current transcoding systems and briefly comment on their pros and cons.

### 2.1. Progressive Data Formats

Traditional (non-progressive) image formats pose a restriction on image transmission: the complete set of data constituting an image has to be transmitted before

the viewer at the receiver's end can see the whole image. With slow Internet connection or high resolution image, the retrieval latency can be in the range of minutes. Progressive image formats improve this situation by allowing a viewer to see an approximated image in its whole without the need to wait for all the data to be received. Early efforts include interlace GIF and progressive JPEG images. Recent leading developments in this area are the advent of JPEG 2000 and MPEG-4.

An interlaced GIF (Graphics Interchange Format) displays images in two passes of alternating lines instead of loading them one line at a time. Depending on which graphics viewer or Web browser is being used, interlaced GIFs may produce a "venetian blind" effect or simply a blurry or blocky image that gradually sharpens. Pages using interlaced GIFs let people see at least the outline of an image sooner; thus the pages often appear to load faster than those with non-interlaced graphics.

Progressive JPEG is developed by the Joint Photographic Experts Group committee [10]; it is no more than a standard JPEG compressed image with the data rearranged into a series of scans of increasing quality. The advantage of progressive JPEG is that if an image is being viewed on-the-fly as it is transmitted over slow network link, one can see an approximation to the whole image very quickly, with gradual improvement of quality as one waits longer; this is much more desirable than a slow top-to-bottom display of the image. The disadvantage is that each scan takes about the same amount of computation to display as a whole regular JPEG would.

With the increasing use of multimedia, image compression requires higher performance as well as new features. JPEG 2000 [5,6,11] is thus being developed to address this need in the specific area of still image encoding. One of the key features of JPEG 2000 is its support for different types of progressiveness when compressing the image. It is possible to have progressive by spatial resolution or progressive by quality. In the former case, the image will increase in size (typically the image grows by a factor of 2 both in height and width) up to its original size, when more bits are received. In the progressive by quality case, the image starts at its original size but the pixel values are very coarsely approximated. As more bits are received the approximation of the pixel values improves. Thus, the receiver side will see an image of initially very poor quality that becomes better as more bits are received. Progressiveness allows the user to stop the transmission when he or she is satisfied with the resolution or quality of the image.

MPEG-4 [13] is the next audio-visual coding standard from ISO after MPEG-1 and MPEG-2. It improves upon the previously successful MPEG standards in areas related to Internet streaming, bandwidth scalability, interactivity, and coding efficiency. In MPEG-4, for video presentation from the same bitstream, the user can extract a sub-bitstream that represents either a lower frame rate, or a lower resolution. This allows the users with lower bandwidth connections to trade-off either motion smoothness or video spatial resolution for a lower bandwidth requirement. The still images coded with MPEG-4 still texture tools support both resolution and quality scalability with very fine granularity. These make MPEG-4 suitable for scalable media delivery over the Internet.

## 2.2. Existing Transcoding Systems

There have been numerous attempts on reducing network bandwidth and/or client latency. In the past few years, there are a growing number of researches on content adaptation. Much work has been done on adapting multimedia objects to bandwidth through transcoding.

Transcoding is a technique employed to dynamically customize multimedia objects for individual client characteristics and prevailing network conditions. Transcoding can be performed along a number of different axes such as color depth, storage space and fidelity, and the specific transcoding technique used depends on the type of multimedia object. Typical examples of transcoding include conversion within media types (e.g., GIF to JPEG conversion), conversion between media types (e.g., video to images), as well as object compression (e.g. lossy compression on a JPEG image). In addition to the ability to achieve content adaptation, transcoding also helps reduce network bandwidth consumption and client latency. Some of the representative transcoding systems are described as below.

Mowser [1,9] is a transcoding proxy that allows a mobile user to specify his viewing preferences, and performs transcoding of HTTP streams accordingly. This system features an extended content negotiation mechanism of HTTP/1.1. On receiving a request from a mobile user, the proxy appends an Accept header to the request, indicating user's preferred representations. In case the server is not able to provide the multimedia content at a resolution appropriate for the mobile user, the proxy transcodes the server response. If the target object is an image, it will be reduced in size or color as

requested by the end user. For videos, representative frames will be selected and transmitted to the user.

In the GloMop model [7,8], the proxy performs on-demand "distillation" of the object received from the server before sending it to the client, so as to deal with client variability and improve end-to-end performance. Distillation is defined as highly lossy, real-time image compression. Based on the client's profile, the image distiller performs one or more of the following operations: size reduction, color quantization and format conversion. In addition, a "refinement" mechanism allows clients to request selected portions at increased quality, possibly the original content. Videos can be distilled along the temporal dimension by limiting the frame rates to a target bit rate. In the spatial domain, operations similar to those used in the image distiller will be performed.

Deployed at the server, the InfoPyramid system [14,16] is proposed to adapt multimedia objects to a wide range of client capabilities through transcoding. Multimedia objects are transcoded offline into multiple resolution (along the dimensions of size, quality, and color depth) and modality (e.g. images can be converted to text, videos to images, text and audio) versions. When the server receives a request client, it determines client capabilities, evaluates the alternative presentations and selects the one delivering the most content value. Caching is also employed in InfoPyramid to eliminate the latency introduced by content selection module. Thus when the system receives a request, it first checks if a client with the same capabilities made a request for the same object previously, and if so, retrieves the cached copy.

To ensure the effectiveness of transcoding, Surendar et al. [2,3] proposed the notion of *quality aware transcoding* in which the efficiency of a transcoding algorithm is defined by the ability to lose more in storage space for a particular loss in information quality. This work is further extended by incorporating the notion of *informed transcoding*, in which images are transcoded only when the estimated time to transmit the original image, given the current estimate of network bandwidth available, is greater than the estimated time to compute the transcoding and the time to send the transcoded image. A hybrid system was implemented and deployed at a proxy server, and the system performance was studied in [4].

Despite the implementation and target difference among these systems, they share the same underlying assumption about transcoding techniques: each content adaptation system completely downloads a target object from a server, transcodes it into a lower quality

version and then sends the transcoded version to the client. Consequently, these approaches have several limitations. Firstly, while the bandwidth consumption over the proxy-to-client link is reduced due to the smaller size of transcoded image, the consumption over the server-to-proxy link remains the same. Secondly, the systems entail significant delay overhead due to the streaming broken down of the streaming transmission of data blocks from server and the nontrivial overhead of the transcoding process. Thirdly, as maintaining multiple variations of an object in the cache is a complicated issue, current systems did not integrate caching mechanism to further improve system performance. Finally, no technique is proposed to reuse data among various transcoded versions of the same object (i.e. building one transcoded version on top of the other transcoded version of the same object). This results in unnecessary consumption of network bandwidth.

### 3. System Architecture for Scalable Pervasive Internet Access with Data Reuse

In view of the limitations that current transcoding systems have (as summarized in the last section), we would like to propose a system architecture for pervasive Internet access. This system should feature efficient real-time transcoding process with maximum data reuse among transcoded object versions and minimum bandwidth usage for the best-fit quality of an object presentation. As a result, it has the following design considerations:

- Proxy centric solution
- On-the-fly transcoding for demanded quality
- Streaming transcoding without data buffering
- Low runtime overhead of transcoding
- Optimized bandwidth usage for the demanded data only.

#### 3.1. Scalable Data Model

With the design considerations discussed in the last section, we propose a scalable data model (SDM) to fulfil streamed transcoding, small transcoding overhead and partial object reuse.

##### 3.1.1. Formal Framework

In our model, a multimedia object is represented by a bit-stream; bit is the elementary unit of an object. An arbitrary sequence of consecutive bits forms a data

block. A data block starting with the first bit is a presentation of an object. We impose several constraints on the data model in order to fulfil requirements such as streamed transcoding, small transcoding overhead and partial object reuse.

In the description of our data model, we adopt the following labels:

- $B$ : Set of all data bits
- $BLK$ : Set of all blocks
- $O$ : Set of all data objects
- $P$ : Set of all presentations

##### Definition 1: Bit Stream

A data object is composed of a continuous stream of bits, denoted as follows:

$$O_n = \text{Union}(b_1b_2b_3 \dots b_{m-1}b_m),$$

where  $O_n \in O$ , and  $b_i \in B$  with  $i \in [0, m]$ .

##### Definition 2: Block

A block is an arbitrary sequence of bits, which can be defined as:

$$blk(i, j) = \text{Union}(b_i b_{i+1} \dots b_{j-1} b_j),$$

where  $blk(i, j) \in BLK$  and  $b_n \in B$  with  $0 < i < j$  and  $n \in [i, j]$ .

There is no constraint imposed on the size of the blocks.

##### Definition 3: Presentation

A presentation of a multimedia object is a variation of the original object. It can be displayed and presented to the user, often at a quality level lower than the original image. In our data model, a presentation is represented as a sequence of data bits, with the only constraint that it must start with the first data bit. A presentation can be expressed using either bits or blocks.

Here we use  $P_i$  to denote a presentation generated by a bit stream of size  $i$ .

$$\text{Bit expression: } P_i = f(b_1b_2b_3 \dots b_{i-1}b_i),$$

where  $b_i \in B$ ,  $i \leq \text{size of original object}$ , and  $f$  represents the encoding function.

Presentation  $P_i$  can also be expressed as a series of blocks, provided that no two blocks overlap one another and the union of these blocks constitute the bit stream from bit 1 to bit  $i$ . Alternatively, it can be expressed in terms of blocks:

Block Expression:  $P_i = f( blk(I_1, I_1) \ blk(I_1+1, I_2) \ blk(I_2+1, I_3) \ \dots \ blk(I_k+1, i) )$

where  $0 < I_1 < I_2 < I_3 \dots < I_k < i-1$ , and  $f$  represents the encoding function.

To achieve efficient real-time transcoding with maximum data reuse and minimum bandwidth consumption, the following properties about the construction of a presentation need to be enforced:

**Property 1: Inclusive Property**

Inclusive property requires that the data constituting a lower quality presentation of an object be contained in a higher quality presentation. Suppose  $P_i$  and  $P_j$  are two presentations originated from the same object, and  $P_j$  is of a higher quality as compared to  $P_i$ , then we have:

$$P_j = F(P_i, blk(i+1, j))$$

where  $i < j$ ,  $blk(i+1, j) \in BLK$ , and  $F$  denotes the encoding function that generate  $P_j$  by using  $P_i$  and  $blk(i+1, j)$ .

**Property 2: Single Pass Property**

According to inclusive property, we have:

$$P_i = F(P_{i-1}, blk(i, i)) \text{ or}$$

$$P_i = F(P_{i-1}, b_i)$$

Similarly,

$$P_{i-1} = F(P_{i-2}, b_{i-1})$$

$$P_{i-2} = F(P_{i-3}, b_{i-2})$$

... ..

$$P_2 = F(P_1, b_2)$$

$$P_1 = F(b_1)$$

The above shows that each data bit is to be encoded without involving data bits following it. This means that the bit-stream is encoded and decoded within a single sequential pass. The single pass property is crucial to streamed transcoding, as it allows a bit-stream to be transcoded without having to wait until an object is downloaded completely.

**Property 3: Union Function F**

We looked into the encoding function  $F$ , and discovered that the choice of union function will lead to minimal transcoding overhead. If  $F$  corresponds to the union function, the transformation of a higher quality presentation into a lower one would be as trivial as filtering out the unneeded data blocks. Similarly the transformation of a lower quality presentation into a high one would be as simple as appending the additional data blocks to the latter. As such,

transcoding of a bit-stream can be achieved without requiring the use of multimedia object decoding and encoding.

Our data model allows fast and direct conversion between different presentations of a multimedia object. Suppose we want to transform between a lower quality presentation  $P_i$  and a higher presentation  $P_j$ . Based on our data model, the following equation holds for transcoding operations:

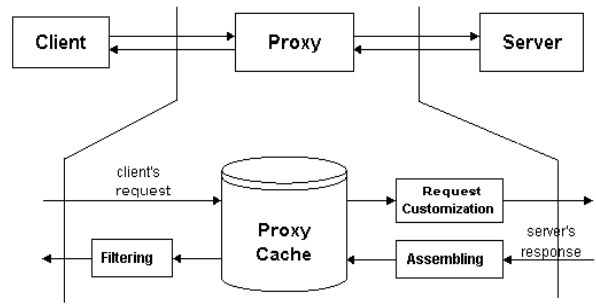
$$P_j = \text{Union}(P_i, blk(i+1, j))$$

**Rules: Data Reuse Among Transcoded Versions of Object**

Either one of these two situations happens when one transcoded version of an object is used to build another version:

- *Conversion from higher quality to lower quality*  
To transform  $P_j$  into  $P_i$  simply discard the data bits ranging from *bit (i+1) to bit j*.
- *Conversion from lower quality to higher quality*  
To transform  $P_i$  into  $P_j$ , we only need to retrieve the data bits ranging from *bit (i+1) to bit j*, and then append them to presentation  $P_i$ .

**3.2. System Architecture Based on SDM**



**Figure 3.1** System Architecture of SDM-Based Transcoding System

The basic architecture of our proxy-based transcoding system architecture is shown in Figure 3.1. Our system deploys a client-side proxy, which is typically far from the remote server. The proxy cache is the core of our system, and it is designed to reap the maximum benefits of data use by extending caching to partial objects. Our system incorporates three modules: **Request Customization**, **Assembling** and **Filtering**. Request customization module determines if a request needs to be customized, and if so, it will customize the request accordingly. When the proxy receives the

server's response, the assembling module will be invoked to concatenate the response with the cached copy so as to build a better quality version. Before transmitting to the clients, the cached content will pass through the filtering module to ensure that the proxy transmits the appropriate version to the clients.

### 3.2.1. Proxy Cache

When the proxy receives a request for a multimedia object, it first checks if a version of the target object exists in the cache. If there is no cached copy, the client's request will be forwarded to the server, and the server's response will be replicated in the proxy cache and transmitted to the client. If the quality of the cached version is below the client's expectation, the request will be modified, since only the missing data needs to be fetched from the server. If the cached version is at a quality level equal to or above client's requirement, the proxy will directly serve the request from its cache.

The proxy cache deployed in our system presents three unique features. They are: (i) caching partial objects, (ii) notion of "partial hit", and (iii) enforcement of single range.

The growing support for partial object retrieval certainly sparks research activities on exploring the benefits of this feature. While there are proxy servers that allow retrieval of partial objects, none of them allow partial objects to be cached. Coupled with the availability of our proposed data model, caching partial objects can significantly reduce client perceived latency and network bandwidth requirement as it maximizes the benefits of data reuse. Caching partial objects ensures that the minimum volume of data is to be transmitted across the network. Hence the caching mechanism employed in our system applies to partial objects as well as the complete objects.

As traditional proxies merely allow complete objects to be cached, an object is either present or absent in the cache. These two statuses are known as *cache hit* and *cache miss* respectively. Now that partial objects are cached as well, an object can be either completely present or partially present in the cache. We introduce the notion of "*partial hit*" to tell it from the traditional "*complete hit*". In response to the three possible cache statuses of the target object, a request will be classified into one of the following categories: complete hit, partial hit, and cache miss.

Extending caching to partial objects leads to an interesting question: shall we enforce single range on

the objects stored in the cache, or rather allow multiple ranges? Allowing multiple ranges means that an object may present itself as an arbitrary list of fragmented ranges, e.g. 2K-6K, 7K-9K, 13K-15K, and so on. This will add substantial complexity to the implementation of the request customization module and assembling module. Besides, overly fragmented ranges tend to lead to performance degradation. Hence we believe that it is necessary to enforce single range constraint on the cached objects. In order for the cached content to be meaningful, we also require that the cached content should be a presentation, i.e. it should start with the first data bit.

### 3.2.2. Request Customization Module

A requests needs to be customized when the cached presentation does not exist or it is not up to the client's expectation. If the request is a *complete hit* (i.e. the requested object is completely available in the cache), the proxy will not forward it to the server, but instead directly respond to the request by transcoding and transmitting the cached presentation to the client. In the following we describe when and how to perform customization on a request that it is not a complete hit.

1. If the request is a *partial hit*, whether customization is needed depends on the situation. Suppose a client requests data in the range  $[r_1, r_2]$ , and the available data range in the cache is  $[1, p]$ .
  - $r_2 \leq p$   
This means that all the requested data are available in the cache. Such requests can be directly served without the need to contact server. Thus customization is not needed in this situation.
  - $r_1 \leq p < r_2$   
This means that part of the requested data is in the cache; the rest are missing. In this case, only the missing data need to be retrieved from the server. Thus the request should be modified to fetch data in the range  $[p+1, r_2]$ , rather than the original range  $[r_1, r_2]$ .
  - $r_1 > p$   
This refers to the case when the requested data are not available in the cache. To fulfil the single range constraint, the request should be modified to retrieve data in the range  $[p+1, r_2]$ .
2. If a request asking for data in the range  $[r_1, r_2]$  is a cache miss, the request should be customized if  $r_1 > 1$ .

- $r_1 = 1$   
If the requested range starts with the first bit, the request will be forwarded to the server without any modification.
- $r_1 > 1$   
Due to the single-range constraint, the request has to be modified to retrieve data in the range  $[1, r_2]$ .

### 3.2.3. Assembling and Filtering Modules

The assembling module is employed to transcode a lower quality presentation into a higher quality presentation. As mentioned in Section 3.3, one big benefit of our proposed data model is that it makes such transcoding as simple as appending additional information to the low quality presentation. In our system, the assembling module simply extracts the data related to object content (that provides more details about the object) from the received server's response and appends them to the corresponding partial object stored in the cache.

The filtering module is used to transcode a higher quality presentation into a lower quality presentation. With the availability of our proposed data model, such transcoding is virtually a filtering operation that discards the unneeded details. In our system, when the quality of the cached presentation is above the client's needs, the proxy will respond to the request by invoking the filtering module to filter out the unneeded information before transmitting the cached object to the clients.

### 3.3. Analysis of SDM-Based System

In a typical proxy-based transcoding system, the proxy's role is to retrieve objects from the servers on the client's behalf, transcode the object to adapt to client's variations, and transmit the transcoded object to the client. In our proposed data model, transcoding of an arbitrary data block does not require knowledge of the following data blocks. This allows the proxy to pipeline the data blocks streamed from the server to the proxy, then to the client. As such, our system preserves the block data streaming with respect to data transfer across the network.

About the content adaptation, our system adapts objects to individual client's characteristics through on-the-fly transcoding. In particular, the assembling module is employed to handle transformation of a

lower quality presentation to a higher quality presentation; the filtering module is designed to handle transformation of a higher quality presentation into a lower quality one. In addition, as the clients are allowed to request for an arbitrary range of data, our system allows precise control over the object quality so as to satisfy the wide spectrum of clients.

Same as any other proxy-based transcoding system, our system minimizes the bandwidth requirement over the proxy-client link by transmitting the customized presentation to the clients. More importantly, our system offers a substantial benefit that is lacking in current systems: it minimizes the bandwidth requirement over the server-proxy link. This is achieved through the following strategies: first, we propose a data model for multimedia object that allows a lower quality version to be transformed to a higher quality version through a simple append operation; second, we extend caching mechanism to partial objects so as to maximize the benefits of data reuse. These two strategies ensure that the volume of data transmitted across the network is kept to the minimum level.

Our system also addresses real-time delivery from a number of aspects. First, our system minimizes the volume of data transmitted across the network. Second, our proposed data model makes it possible that conversion between different quality presentations does not incur transcoding overhead. Third, as our data model allows streamed transcoding, our system preserves the block data streaming with respect to network data transfer. The above strategies ensure real-time delivery to the clients.

Hence, we see that the capabilities of our system model match the system requirements mentioned previously.

## 4. Sample System Implementation on JPEG2000

In this section, we describe how we implemented our system to support JPEG2000 in detail. We first provide background information on JPEG2000 and HTTP/1.1 protocol – the key technical developments underlying our system. Next, we examine all the feasible approaches, and present our architecture decision.

### 4.1. Background

Before we go into the discussion on the implementation details, it would be helpful to provide

background information on JPEG2000 format and HTTP/1.1 protocol, as they lay down the foundation of our system. JPEG2000, the emerging next generation digital imaging standard, is fully compliant with our proposed data format. It offers a host of features that are unavailable with conventional images such as JPEG and GIF. Some of the demonstrative JPEG2000 bit-streams are available in the JPEG group's official website [11]. We use these bit-streams as the test data for our system. As to HTTP/1.1 protocol, it allows for partial object retrieval, making it possible to take advantage of the scalability feature offered by JPEG2000. Details about these two technical developments are described in the following two subsections.

#### 4.1.1. JPEG2000

With the increasing use of multimedia technologies, image compression requires higher performance as well as new features. To address this need in the specific area of still image encoding, a new standard is currently being developed, the JPEG 2000. It offers features and functionalities that current standards can either not address efficiently or in many cases cannot address at all. Some of its representative features such as progressive transmission by accuracy and resolution, robustness to the presence of bit-errors and random code-stream access, are specially designed to address the requirements of Internet application.

The main attractiveness of JPEG2000 lies in its ability to have scalability in both image quality and resolution. JPEG2000 allows encoders to flexibly arrange the image file to best suit the way the users will need to access the information. For instance an image encoded in a progressive-by-resolution structure, when transmitted across the network, allows the users to see a thumbnail image first, and the image will gradually increase in size (typically the image grows by a factor of 2 both in height and width) till up to its original size. On the other hand, a progressive-by-quality structure begins with a full spatial-resolution version yet with very coarsely approximated pixel values, followed by information that incrementally enhances the accuracy of the pixel values. Although a JPEG2000 bit-stream can be stored in various desired orders such as progressive-by-resolution and progressive-by-quality, it can only exist in one order at a time.

At the same time, JPEG2000 offers great flexibility on a decoder, allowing an end user — the person receiving an image — to choose interactively, in real-time, the resolution and compression. Depending on bandwidth or time limits, users can request a thumbnail, full-

monitor resolution or print-quality resolution. Or a user can simply ask to access the image only at the appropriate resolution for his or her display. If an image is encoded in progressive-by-quality order, rendering a low quality version is as simple as truncating the original bit-stream, without requiring any processing at all. Similarly, if an image is encoded in progressive-by-resolution order, a low-resolution version can be derived by truncating the original bit-stream. Another interesting benefit of JPEG2000 is that change of the progression order of a bit-stream (e.g. change between progressive-by-resolution and progressive-by-quality order) is as trivial as a copy operation with respect to operation complexity.

The impact of JPEG2000 on the Internet is enormous. From the content provider's perspective, JPEG2000 enables them to offer a wide diversity of resolutions and quality levels for an image with the same bit-stream, thus relieves them from maintaining multiple copies for each image. From the client's perspective, JPEG2000 gives them fine-grained control over image quality and spatial resolution, depending on their needs and Internet connectivity speed. More importantly, JPEG2000 allows for accumulative transmission of an image object. As mentioned earlier, JPEG2000 organizes data in such a way that basic layer is presented first, and the following refinement layers serve to enhance the quality or resolution level of the previous layer. If a client has previously downloaded a low quality version, he only needs to request for a delivery of the additional refinement layers, rather than all the involved layers, in order to render a higher resolution version. As such, by downloading a few extra layers at a time, the client will get to view the image at increasing quality level.

We exploit the benefits of accumulative transmission in our proxy-based transcoding system by supporting caching of partial objects. A partial object refers to an object that is not in its entirety. In other words, a partial object is part of a complete object. A low quality version of a JPEG2000 image is an example of partial object. With the proxy's support for partial object caching, the content of a low quality copy that has been fetched from the server previously can be shared among a group of clients connected to the same proxy. Allowing caching of partial objects maximizes the benefits of data reuse, and thus leads to minimum end-user latency and bandwidth requirement.

#### 4.1.2. HTTP/1.1

HTTP/1.1 protocol allows a client to request portions of an object via *Range* requests. The only range unit

defined by HTTP/1.1 is "bytes". In other words, HTTP/1.1 supports only ranges of bytes, although the range mechanism is extensible to other units (such as chapters of a document, or frames of a movie). In HTTP/1.1, a client makes a range request by including the *Range* header in its request, specifying one or more contiguous ranges of bytes. If the server supports range request, it returns one or more ranges in the response, and indicates the offset and length of the returned range using a *Content-Range* header. If the server does not support range request, it will ignore the Range header in the received request. We describe the Range and Content-Range headers as follows.

### A. Byte Ranges Specifier

In an HTTP request, byte-ranges-specifier is used to describe the range of bytes to be retrieved. It is defined by HTTP/1.1 in the following form:

bytes=X-Y

where:

- X is the byte offset of the first byte in a range.
- Y is the byte offset of the last byte in the range.

Description

- Both byte positions X and Y specified are inclusive.
- Byte offsets start at zero.
- One of the byte offsets may be missing, but both of them cannot at the same time.
- If X value is absent, X is taken to be zero.
- If Y value is absent, Y is taken to be the end of the object file.

Examples of byte-ranges-specifier values (assuming an object of length 10000):

- The first 500 bytes (byte offsets 0-499):  
bytes=0-499
- The second 500 bytes (byte offsets 500-999):  
bytes=500-999
- The final 500 bytes (byte offsets 9500-9999):  
bytes=-500 or bytes=9500-

### B. Range Header

HTTP retrieval requests using GET methods may request ranges of the object by using the Range header. The Range header consists of the keyword "Range", followed by a colon, and ended with a byte-ranges-specifier. That is,

Range: bytes=X-Y

An example of an HTTP GET request containing a Range header is as follows:

```
GET http://www.nus.edu.sg Range: bytes=2500-4000
```

### C. Content-Range Header

If the server supports ranges, the response to a range request carries a status code "206", and contains a Content-Range header indicating the offset and length of the returned range along with a Content-Length header showing the number of bytes actually transferred. An instance of a partial object response is as follows:

```
HTTP/1.1 206 Partial Content
Date: Thu, 03 May 2001 09:22:12 GMT
Last-Modified: Thu, 03 May 2001 07:12:27 GMT
Content-Range: bytes 2500-4000/7614
Content-Length: 1501
```

A Content-Range header is in the following form:

Content-Range: bytes X-Y/Z

where:

- X is the byte offset of the first byte returned.
- Y is the byte offset of the last byte returned.
- Z is the total size of the object file in bytes.

HTTP/1.1 also requires that if a proxy that supports ranges receives a Range request, forwards the request to the origin server, and receives an entire object in reply (this suggests that the origin server does not support ranges), it should only return the requested range to its client.

## 4.2. Feasible Approaches and Decisions

With the availability of JPEG2000, it makes sense to request and retrieve partial objects. Suppose both the client and the server are HTTP 1.1 compliant, the proxy has a number of options when it comes to how to handle partial object requests. The list of feasible options differ primarily in three aspects:

- Shall we allow caching of partial objects?
- Shall single range requirement be enforced on cached partial objects?
- How to handle a partial object request when the target object is partially available in the proxy cache?

### 4.2.1. Shall We Allow Caching of Partial Objects?

It has been shown that caching mechanism can effectively reduce client perceived latency and bandwidth requirement by replicating web objects at a place near the clients. When the same object is referenced again, it will be directly retrieved from the

cache. Current proxy-based systems incorporate caching mechanism with the assumption that the integrity of the object must be preserved. That is, an object is either present or not in the cache, but cannot be *available in part*. As only complete objects are to be cached, a partial object request that is a cache miss might be handled in one of the following ways:

**Approach 1:** The proxy retrieves the whole object from origin server. The entire object is then cached in the proxy. However, the proxy does not return the entire object to the client, but instead transmits only the portion that is requested by the client.

*Pros and cons:* This approach is simple to implement. However, if the server is across slow line, and a client requests for only a small proportion, say one quarter, of an object, it is not a desirable solution because considerable network bandwidth are wasted over unneeded data.

**Approach 2:** Apart from the original client request, the proxy creates another partial object request to prefetch the remaining data portion. Both requests are sent to the origin server. The reply to the original request will be forwarded to the client. The object content embedded in the two replies will be assembled together and stored in the proxy cache as a complete object.

*Pros and cons:* Like approach 1, this approach is not economical when the server is across slow line. Moreover, if the two requests are simultaneously sent to the server, it is difficult to properly assemble the content because data blocks of two HTTP replies may arrive in an interleaved fashion. To simplify the assembling, the new request should not be sent to the server until the reply to the original request returns completely.

The above approaches do not benefit from the scalability feature of JPEG2000, as the proxy still has to download objects completely. To make the best of the advent of progressive data formats, caching mechanism should be extended to partial objects as well. With the support for caching of partial objects, the proxy will handle a cache-miss partial object request as follows:

**Approach 3:** When such a request arrives, the proxy directly forwards it to the server. The server's response, which contains a partial object, will be replicated into the cache and then forwarded to the client.

*Pros and cons:* This approach minimized network bandwidth consumption. It is desirable when the server is across a slow line, and the object size is bulky.

However, this approach calls for more complicated mechanisms to manage and maintain the partial objects stored in the cache.

In the real-world Internet environment, the network line between the proxy and server is typically slow and congested. Thus we adopt approach 3 as it best addresses this situation.

#### 4.2.2. Shall Single Range be Enforced on Cached Partial Objects?

Consider this scenario: When the proxy receives a partial request asking for data bytes in the range from 8K to 10K, it looks up the cache for the requested object and discovers that the object is partial available with data bytes ranging from 1 to 6K. An interesting question would be: shall we enforce single range on the cached partial objects? If so, the proxy will request data ranging from 6K+1 to 10K instead. Otherwise, the proxy simply forwards the original request, which asks for data in the range from 8K to 10K, to the server.

If the proxy does not enforce single range property, the cached object likely contains multiple ranges. For example, a cached object may appear as a list of ranges: 2K-6K, 7K-9K, 13K-15K, and so on. If a client requests for data bytes ranging from 1K to 14K, the proxy has to create three requests asking for data bytes in the range 1K to (2K-1), (6K+1) to (7K-1), and (9K+1) to (13K-1) respectively. This apparently leads to implementation difficulty and performance deterioration of the system. Hence we impose single range requirement on cached partial objects in our system.

#### 4.2.3. How to Handle Subsequent Partial Object Requests?

Cache content may get replaced to make room for newly fetched objects, as the cache storage is finite and limited. As such, the client copy may differ from the cached copy in the proxy. The relation between the requested range and available data range in the cache can be categorized into three types, namely subset, overlapping or non-overlapping.

##### *Case A: Subset*

The requested data range is a subset of the available data range of the cached copy. That is, all the requested data are available in the cache. In this case, the proxy can immediately service such requests without the need to contact server.

### Case B: Overlapping

The byte range of requested data overlaps with that of the cached copy. In other words, part of the requested data is available in the cache whereas the rest is not. The proxy may handle this scenario using one of the following approaches:

**Approach 1:** The proxy simply forwards the request to the origin server, and then directly transmits the server's response to the client. But before appending the response to the cached copy, the proxy has to filter out the redundant information, i.e. the overlapping portion of the request and the cached copy.

**Approach 2:** The proxy modifies the request to fetch only the missing data portion. The object content embedded in the server's response will be appended to the cached copy. The proxy transmits the requested data portion to the client.

Approach 2 has several advantages over approach 1. First, it consumes less network bandwidth as it fetches only the missing data portion. Second, the client-perceived response time is minimal as the overlapping data portion is retrieved from the client-side proxy, instead of remote server. Last, the filtering process is avoided as only the missing portion is to be retrieved. Hence, approach 2 is our choice.

### Case C: Non-Overlapping

There is no intersection between the requested data range and the available data range in the cache. This situation arises when the client copy is at a quality level identical to or higher than that in the cache. In this case, all the requested data is to be retrieved from origin server. Three possible approaches are presented in the following:

**Approach 1:** The proxy forwards the original request to the server. Then the server's reply will be passed on to the client and appended to the cached copy.

**Approach 2:** The proxy creates another request for the intermediate portion, which is available in the client side but missing in the cache. Both requests are to be sent to the server. The reply to the original request will be forwarded to the client. The object content embedded in both replies is to be appended to the cached copy.

**Approach 3:** The proxy modifies the request to fetch the intermediate range as well as the requested range. The server's response will be appended to the cached copy. The data in the intermediate range will be filtered out before the proxy forwards the server's response to the client.

Approach 1 contradicts single range requirement. Approach 2 leads to implementation difficulty and incurs more processing overhead as compared to approach 3. Approach 3 is simple, and it fulfills the single range requirement imposed on cached object. Hence, we choose approach 3.

In conclusion, our approach allows partial objects to be cached in order to minimize network bandwidth consumption and user perceived latency. Single range requirement is enforced in our system by customizing original requests when necessary.

## 4.3. Illustration of Scalable Transcoding Effect of JPEG2000



Figure 4.1. Various Transcoded Versions of "lena" Image

We implemented our SDM and system model of JPEG2000 on SQUID to study its scalable transcoding effect. We downloaded JPEG2000 demonstrative bit-streams and decoder from Neximage website [12]. In order to view these image objects, we use the decoder to convert the JPEG2000 bit-streams into JPEG

images. Figure 4.1 displays the ‘lena’ image at original quality level. We truncate the original JPEG2000 bit-stream at certain points, and decode the bit-stream before the cut-off point. Figure 4.1(a) to Figure 4.1(f) display the bit-streams with cut-off points at a position proportional to 1/16, 1/8, 1/4, 1/2, 3/4 of the original bit-stream respectively.

From the above pictures, we observe that the first four layers (see Figure 4.1(d)) can render a presentation at a reasonably good quality. In addition, fetching the next four layers and appending them to the first four layers can generate a better-quality image as shown in Figure 4.1(e). We can also see from Figure 4.1(f) that the first 12 layers can deliver a presentation almost identical to the original bit-stream. These pictures demonstrate the scalability feature of JPEG2000 bit-stream.

#### 4.4. Mathematical Analysis on Performance Benefit

In this section, we would like to perform a mathematical analysis on the performance benefit of our model over conventional transcoding systems.

##### Conventional Model

As illustrated in Figure 4.2, conventional proxy retrieves the entire image object of size  $S$  over the server-proxy connection with effective bandwidth  $B_{sp}$ . Once a data block arrives, the proxy transfers it to client over the proxy-client connection having effective bandwidth  $B_{pc}$ .

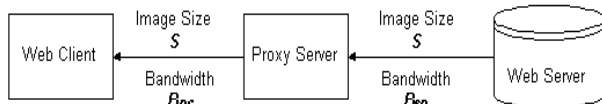


Figure 4.2. Model of Conventional Transcoding Proxy.

Figure 4.3 depicts the timeline corresponding to an HTTP transaction between a client, traditional proxy and a server. Packets 1, 2 and 3 correspond to the SYN, SYN/ACK and ACK packets that are exchanged as part of the TCP 3-way handshake in the connection establishment phase between the client and the proxy. The client sees this connection as established after a delay of approximately  $RTT_{pc}$  (round trip time between proxy and client), which is the network roundtrip latency between the client and the proxy. At this point the client sends packet 4 (at approximately time  $1.5 * RTT_{pc}$ ), which contains the HTTP request. After the server proxy gets this packet, it initiates a connection with the server (packets 5, 6, 7). The latter connection is established after a further delay of  $RTT_{sp}$ , which is

the roundtrip latency of the link between the server and the proxy. The proxy then forwards the client’s HTTP request to the server (packet 8).

On receipt of this HTTP request, the server prepares the data (packet 9) and then sends it off. This takes another  $L = 0.5RTT_{pc} + 0.5RTT_{sp}$  time before the client gets to see it. The amount of time  $T_{transfer}$  that it takes to transfer the response is dependent on the size of the data and network bandwidth.

Thus the client-perceived response time can be expressed as the sum of the following three terms:

$$\text{Latency} = 2RTT_{pc} + 2RTT_{sp} + \max(S/B_{pc}, S/B_{sp})$$

Since the assumption is that  $B_{pc}$  is faster than  $B_{sp}$ , the above formula can be simplified as follows:

$$\text{Latency} = 2RTT_{pc} + 2RTT_{sp} + S/B_{sp} \quad (1)$$

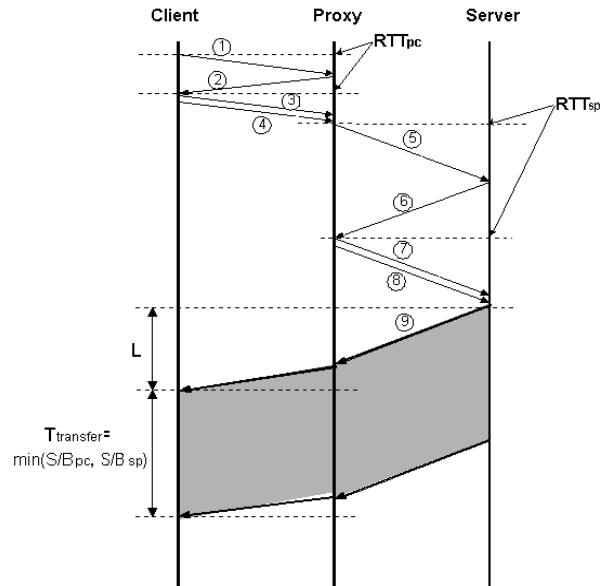


Figure 4.3: Timeline for a HTTP Request in a Conventional Proxy.

##### Our Improved Model

As discussed earlier, our system model allows partial objects to be cached. Suppose the first  $S_c$  bytes of the image are available in the proxy. Then the proxy only needs to request the server for the unavailable data portion of the image. As shown in Figure 4.4, the proxy fetches  $S - S_c$  bytes of data from the server, but transfers  $S$  bytes of data to the client.

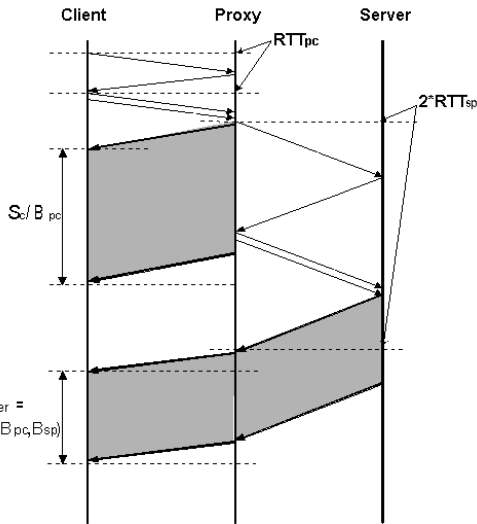


**Figure 4.4.** Model of Our Improved Proxy.

We describe how to calculate the client-perceived response time in the following:

*Case I:*  $2RTT_{sp} > 0.5RTT_{pc} + Sc/B_{pc}$

In the situation when  $2RTT_{sp} > 0.5RTT_{pc} + Sc/B_{pc}$ , the proxy finishes transmitting the cached portion of the object before receiving the first packet of response from server. This situation usually occurs when the proxy-server connection takes long time to establish and the size of cached portion is relatively insignificant as compared to that of the unavailable portion.



**Figure 4.5:** Timeline for a HTTP Request in Our Improved Model (Case I)

Figure 4.5 depicts the timeline of a partial object request under such circumstance. The client-perceived response time equals that of retrieving an object of size  $S-S_c$  from server. That is,

$$\begin{aligned} \text{Latency} &= 2RTT_{pc} + 2RTT_{sp} + \max((S-S_c)/B_{pc}, (S-S_c)/B_{sp}) \\ &= 2RTT_{pc} + 2RTT_{sp} + (S-S_c)/B_{sp} \end{aligned} \quad (2)$$

*Case II:*  $2RTT_{sp} < 0.5RTT_{pc} + Sc/B_{pc}$

In this case, the proxy starts receiving response from the server before it finishes transmitting the cached portion to the client. This often occurs when the connection set-up latency between server and proxy is small and the cached data portion is considerably large.

Figure 4.6 depicts the timeline corresponding to a partial request under such circumstance. The client-perceived response time in this case is as follows:

$$\text{Latency} = 2RTT_{pc} + \max(S/B_{pc}, 2RTT_{sp} + (S-S_c)/B_{sp})$$

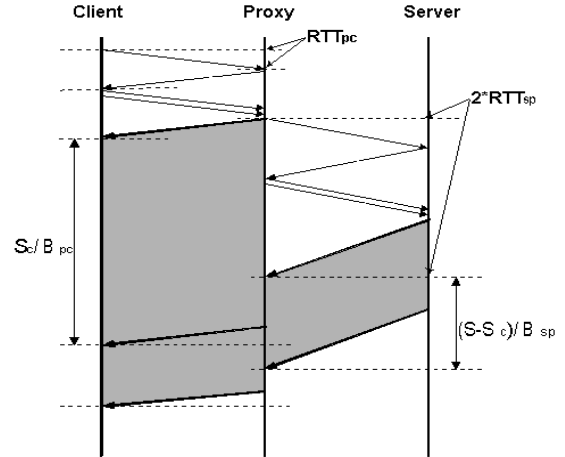
That is, if  $S/B_{pc} < 2RTT_{sp} + (S-S_c)/B_{sp}$ , then

$$\begin{aligned} \text{Latency} &= 2RTT_{pc} + 2RTT_{sp} + (S-S_c)/B_{sp} \quad \text{Same as (2)} \end{aligned}$$

Otherwise,

$$\text{Latency} = 2RTT_{pc} + S/B_{pc} \quad (3)$$

Formula (3) only applies in a small percentage of the cases, in which the connection over server-proxy link is rather fast and a large fraction of the requested content are already available in proxy cache. Formula (2) applies to the majority of the cases.



**Figure 4.6:** Timeline for a HTTP Request in Our Modified Model (Case II)

To evaluate the performance benefits of introducing our modified proxy system, let's look at the ratio of transmission latency in the original system to that in the modified system. As the round trip time i.e.  $2RTT_{pc}$  and  $2RTT_{sp}$  are negligible as compared to the data transfer time, the ratio of formula (2) over (1):

$$\frac{2RTT_{pc} + 2RTT_{sp} + (S-S_c)/B_{sp}}{2RTT_{pc} + 2RTT_{sp} + S/B_{sp}} \approx \frac{S-S_c}{S}$$

This indicates that the ratio of transmission time approximates the ratio of the size of unavailable data portion vs. that of the entire object. The upper bound of performance benefit is the ratio of formula (3) over (1):

$$\frac{2RTT_{pc} + S/B_{pc}}{2RTT_{pc} + 2RTT_{sp} + S/B_{sp}}$$

## 5. Conclusion

In this paper, we presented a scalable multimedia data model and its supporting proxy architecture system to support reusable pervasive Internet access. It offers the following benefits: ability to adapt multimedia object to individual client's need, minimum bandwidth requirement and real-time delivery. We further illustrate the feasibility and capability of our model and architecture through our support to JPEG2000 on SQUID proxy server system. The system takes advantage of two recent technical developments – JPEG2000 data format and HTTP/1.1 protocol. The former is an emerging digital imaging standard that organizes data as a succession of layers, with each layer being one quality increment to its previous layer. The latter allows the clients to retrieve portions of the object via Range method. We implemented our system on top of the freely available Squid proxy server system.

Compared to previous real-time transcoding systems, we make the following specific contributions. Firstly, our SDM-based system dynamically adapts an object to individual client's characteristics in real-time through on-the-fly transcoding. With the property of progressiveness in our SDM, the transcoding process hardly requires any computation, and thus it incurs minimal overhead. More importantly, due to the single pass property of SDM, our transcoding technique is distinguished from those employed in current systems in that it successfully preserves the streaming nature of HTTP connections. This feature is crucial to achieving real-time delivery. Secondly, unlike most proxy-based transcoding systems, there is substantial saving in serve-proxy bandwidth, which is usually the network bottleneck of web retrieval. Thirdly, we introduce the notion of partial hit to maximize the potentials of data reuse from one transcoded version of an object to the next. This is achieved through the inclusive property for presentation and the union function property for transcoding. With maximum data reuse, it gives to an important feature of cumulative browsing. For a client with slow connection to the Internet, he may wish to download a low quality version of the object. With the support from SDM, this is as trivial as fetching the first several layers of the object bit-stream. The next time a client (who might or might not be the same as the first one) wants to view a better quality version, he only needs to fetch subsequent refinement layers and assemble them with the lower quality version he

already has. Consequently, as the visit frequency of an object increases, the visual effect of the presentation improves until it is exactly the same as the original object.

## References

- [1] H. Bharadvaj, A. Joshi and S. Auephanwiriyakul. An active transcoding proxy to support mobile Web access. *Proceedings of 17<sup>th</sup> IEEE Symposium on Reliable Distributed Systems*, October 1998.
- [2] S. Chandra and C. S. Ellis. JPEG compression metric as a quality aware image transcoding. *Proceedings of USENIX 2nd Symposium on Internet Technology and Systems*, page 81-92, Boulder, CO, October 1999.
- [3] S. Chandra, C. S. Ellis and A. Vahdat. Differentiated multimedia Web services using quality aware transcoding. *Proceedings of INFOCOM 2000 - Nineteenth Annual Joint Conference of the IEEE Computer And Communications Societies*, 2000.
- [4] S. Chandra. Quality aware transcoding: an application level technique to dynamically adapt multimedia content. Ph.D. thesis, Computer Science, Duke University, Durham, NC, 2000.
- [5] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 still image coding system: an overview. *IEEE Transactions on Consumer Electronics*, Vol 46, No. 4, pp. 1103-1127, November 2000.
- [6] D. S. Cruz and T. Ebrahimi. An analytical study of JPEG2000 Functionalities. *Proceedings of IEEE International Conference on Image Processing*. September 2000.
- [7] A. Fox and E. A. Brewer. Reducing WWW latency and bandwidth requirements by real-time distillation. *Proceedings of the 5th International World Wide Web Conference*, Paris, France, May 1996.
- [8] A. Fox, S. D. Gribble, E. A. Brewer and E. Amir. Adapting to network and client variability via on-demand dynamic Distillation. *Proceedings of ASPLOS-VII*, Cambridge, MA, October 1996.
- [9] A. Joshi. On proxy agents, mobility, and Web access. In *ACM/Baltzer Journal of Mobile Networks and Nomadic Applications (MONET)*, December, 2000.
- [10] The JPEG group's official homepage.  
URL: <http://www.jpeg.org>.
- [11] JPEG 2000 WhitePaper prepared by Digital Imaging Group. JPEG2000 offers new opportunities to enrich image content and applications flexibility.  
URL: <http://www.ecs.soton.ac.uk/~km/docs/jpeg2000.doc>

- [12] Demonstration of JPEG2000 bit-streams and decoder.  
URL: <http://itswww.epfl.ch/~neximage/>.
- [13] J. Liang. New trends in multimedia standards: MPEG4 and JPEG2000. *Informing Science Special Issue on Multimedia Informing Technologies* – part 1, Volume 2, No 4, 1999.  
URL: <http://citeseer.nj.nec.com/talluri97robust.html>
- [14] R. Mohan, J. R. Smith and C. S. Li. Adapting multimedia Internet content for universal access. *IEEE Transactions on Multimedia*, 1(1):104--114, March 1999.
- [15] A. Ortega, F. Carignano, S. Aver and M. Vetterli. Soft caching: Web cache management techniques for images. *IEEE Signal Processing Society 1997 Workshop on Multimedia Signal Processing*, June 1997.
- [16] J. R. Smith, R. Mohan and C. S. Li. Transcoding Internet content for heterogeneous client devices. *Proceedings of IEEE International Conference on Circuits and System*. May, 1998.