

# Versatile Transcoding Proxy for Internet Content Adaptation

Jung-Lee Hsiao, Hao-Ping Hung<sup>+</sup> and Ming-Syan Chen\*  
Department of Electrical Engineering  
National Taiwan University

<sup>+</sup>Graduate Institute of Communication Engineering  
National Taiwan University

E-mail: mschen@cc.ee.ntu.edu.tw, {jlhsiao, hphung}@arbor.ee.ntu.edu.tw

## Abstract

Recent technology advances in multimedia communication have ushered in a new era of personal communication. Users can ubiquitously access the Internet via various mobile devices. For the mobile devices featured with lower-bandwidth network connectivity, transcoding can be used to reduce the object size by lowering the quality of a multimedia object. A transcoding proxy is a proxy capable of converting a multimedia object from one form to another, which trades object fidelity for size. In view of the monolithic transcoders which only provide transcoding services and have limited performances due to the unknown data types and protocols in the prior research, we propose the architecture of Versatile Transcoding Proxy, denoted by VTP. Based on the concept of the agent system, the VTP 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. In order to enhance the effectiveness of the VTP architecture, we adopt the concept of dynamic cache categories and propose a scheme DCC-MPR, standing for Maximum Profit Replacement with Dynamic Cache Categories. In scheme DCC-MPR, fine granularity control in the number of cache categories is achieved by building a *weighted transcoding graph* which depicts the transcoding relationship among transcodable versions dynamically. Based on the *weighted transcoding graph*, scheme DCC-MPR performs cache replacement according to the content in the *caching candidate set*, which is generated by the concept of *dynamic programming*. The experimental results show that the proposed architecture VTP and the corresponding scheme DCC-MPR have better performances in many aspects compared to the conventional transcoding proxy systems.

---

\*The corresponding author

# 1 Introduction

With the revolution of mobile communication technology, users can access the Internet anywhere, any time, with heterogeneous mobile devices, such as handheld PCs, personal digital assistants (PDAs), and WAP-enabled cellular phones [11]. These devices are different from one another in their computing capability, network connectivity and screen size. Also, due to the limited bandwidth of mobile communication, the traditional content of a web object for desktop computers might not be suitable for a mobile device. Hence, there is a desire to transcode the content to a degraded format that is more appropriate to be presented on the mobile devices [9][20].

In general, as a lossy process, the transcoding technology converts a multimedia object from one form to another, which trades object fidelity for size and preserves the important information. A proper transcoding process will effectively reduce the transmission time of a web object. However, a transcoding system should strike a compromise between the data information and the transmission efficiency. If there is too little distillation, the time spent in transmission will probably be still long. In contrast, with too much distillation, useful information residing in the message might be lost [17]. Moreover, transcoding is useful when the device does not have the capability of processing the content. e.g., for a text-only mobile device, we can transcode images into a text file before sending the content to the device.

As mentioned in [6], transcoding systems can be divided into three classes according to the location where the transcoding process takes places, i.e., the client-based, the server-based, and the proxy-based transcoding systems. It is noted that transcoding at client side is costly due to the limitation in both bandwidth and computing power. On the other hand, transcoding at server side is not flexible to satisfy the client's requirement and will require too much unnecessary storage. Thus, the transcoding system is often implemented at an intermediate proxy [4][17].

Conventional transcoding proxies can be further divided into two different classes according to the way how the transformation logic is applied. The first class is referred to as the *fixed transcoding proxy*, as shown in Figure 1, where the transcoding proxy merely transcodes the input into the output without any context-aware processing. An example of this kind is the HTML-WML translator [25]. The advantageous features of this kind of transcoding proxy are the simplicity of the structure and the robustness of the system. However, such a transcoding proxy system lacks flexibility and may not be able to support various demands of heterogeneous client devices.

The second class is referred to as the *heuristic transcoding proxy*, as shown in Figure 2. The transcoding proxy is able to read the capability profile from the client device and tries to transform the content according to the device capability [15][19]. However, since the proxy does not have any knowledge about which information is important, it is difficult to determine the transformation strategy of the content. Moreover, the heuristics adopted by the transcoding proxy system tend to

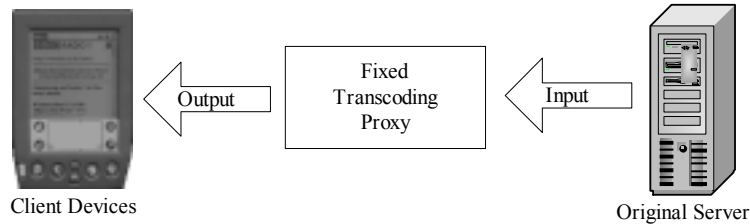


Figure 1: Schematic of a fixed transcoding proxy

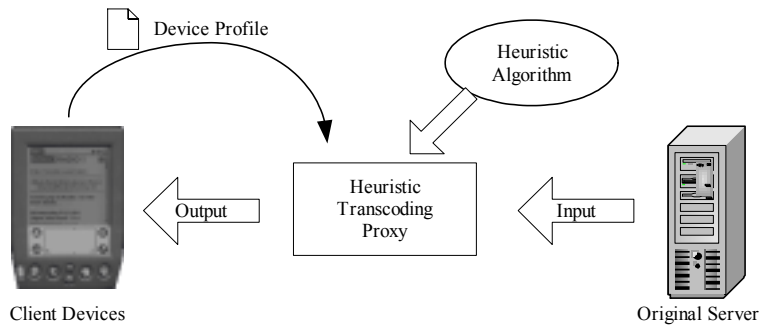


Figure 2: Schematic of a heuristic transcoding proxy

induce unpredictable results. Although recent researchers have proposed various heuristics [7][8], these heuristics still suffer from the loss of important information or the loss of opportunities for better transcoding. The work in [17] has recognized this issue, and proposed a server directed transcoding architecture to remedy the problem. However, this work needs to make an extension to the current HTTP protocol, and its practicability may thus need further justifications.

There are many ways to implement a transcoding proxy system. However, most of them are *monolithic transcoders*, meaning that they can only provide transcoding services to the content type or the protocol which can be recognized in advance. When there is a desire to deal with a new content type or a communication protocol, the upgrade of the whole architecture is inevitable. This consideration makes it rather difficult to maintain the transcoding proxy system.

Consequently, to remedy the above problems, we propose in this paper the architecture of Versatile Transcoding Proxy (denoted by VTP) for Internet content adaptation, and have explicitly implemented a prototype to empirically validate the proposed concept. In our framework, the proxy 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 so that the proxy server can avoid the uncertainty of the heuristic transcoding proxy. In addition, the proposed system architecture is designed to provide the capability of supporting arbitrary type of data by

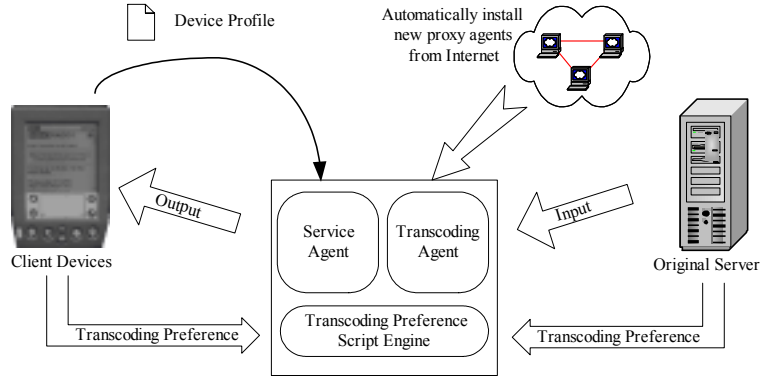


Figure 3: Schematic of the versatile transcoding proxy

utilizing the well-defined software modules in which the new plug-in components can be automatically downloaded from the Internet. It is noted that the VTP architecture can also be used to transcode many types of client-server systems, without being limited to Web contents with HTTP protocol. With the registration of the standard CC/PP device capability profile [24] and built-in CC/PP parser<sup>1</sup>, the proposed versatile transcoding proxy realizes context awareness. Therefore, it is adaptable to heterogeneous devices.

Figure 3 shows the system blocks of the VTP architecture. There are three primary components: the *service agent*, the *transcoding agent*, and the *transcoding preference script*. Service agents are used to abstract the protocol details from the proxy system, while transcoding agents are used to abstract the transcoding actions. The service agent and the transcoding agent enable the VTP system to be adaptable to heterogeneous protocols and data types. On the other hand, since the transcoding preference script, denoted by TPS, is developed in the JavaScript language [1], the VTP system is programmable so that the user or the content provider can configure the related methods when a web object is transcoded. This flexible design of VTP is intrinsically different from other proxy implementations, and as will be empirically demonstrated later, is in fact an advantageous feature of VTP. The TPS can also be embedded in the original data stream, e.g., to enable the server to transmit the transcoding preference to the proxy.

Moreover, to enhance the effectiveness of the VTP architecture, a transcoding scheme DCC-MPR is proposed to maintain the cached objects and perform cache replacement in the transcoding proxy. Two important components are contained in scheme DCC-MPR: mechanism DCC (standing for Dynamic Cache Categories) and algorithm MPR (standing for Maximum Profit Replacement).

<sup>1</sup>VTP adopts the CC/PP parser of the “delicon” open source project to deal with the CC/PP profile. The related information of “delicon” project can be found in <http://sourceforge.net/projects/delicon/>. Since the vocabulary is clearly defined in CC/PP, there is no semantic interoperability mechanism in VTP.

Scheme DCC offers fine granularity control in the number of cache categories by building a *weighted transcoding graph* which depicts the transcoding relationship among transcodable versions dynamically. From the transcoding relationship among categories, algorithm MPR performs cache replacement according to the content in the *caching candidate set*, which is generated by the concept of *dynamic programming*. Algorithm MPR can be divided into two phases. The first phase performs when the proxy has sufficient space. In this phase, once the object is queried, it will be cached to increase the *profit* for future access. The second phase performs when the proxy has insufficient space. In this phase, the cache replacement is performed according to the priority of the requested object.

In order to validate the feasibility of the VTP architecture, we implement a prototype of the VTP architecture and conduct several experiments. First, we measure the performance of the VTP architecture by using an image distillation transcoding agent to transcode some JPEG images for different device profiles. Moreover, we investigate the performance of email services under the VTP architecture by using a POP3 service agent. We also measure the energy consumption of the iPAQ PDA when a user browses the images on Internet to show that VTP performs well in reducing energy consumption. To verify the effectiveness of scheme DCC-MPR, we compare the proposed scheme to algorithm AE, standing for Aggregate Effect [10], which is a cache replacement algorithm used for the transcoding proxy system. By measuring the memory footprint of the VTP implementation, it is empirically shown that the footprint of VTP is rather low compared to other transcoding proxy systems. In addition to the content transcoding, the VTP framework is also very useful in content filtering. In the final experiment, we develop an adult webpage block service as an example. Under the VTP framework, a special transcoding agent capable of scanning the content and blocking the adult Web sites is also implemented to demonstrate the flexibility of this VTP architecture. From the above experiments, it is shown that the VTP architecture is a feasible and flexible framework in the transcoding proxy system.

The rest of this paper is organized as follows. In Section 2, we introduce some related previous works. In Section 3, we present the details of designing the VTP architecture, and discuss the advantageous features and the usefulness in the server transcoding preference application. In Section 4, design of scheme DCC-MPR is described. In Section 5, the performances of the VTP architecture are evaluated via the empirical studies. Finally, this paper concludes with Section 6.

## 2 PRELIMINARIES

### 2.1 Related Work

TranSquid [19] is a transcoding proxy system based on the famous Internet HTTP proxy server - squid. Compared to the other systems which can only cache the original versions of the web object,

the sub-system of this architecture is capable of caching the transcoded versions. The key idea in [19] is to divide the cache entries into three predefined cache categories according to the device capability. In contrast, with the transcoding agent, the VTP architecture designed in this paper allows the number of cache categories to change dynamically in order to adapt to the heterogeneous characteristics of different data types.

The Web Intermediaries (WBI) [3][18] is part of the IBM pervasive computing technology. A set of APIs for writing Java-based plug-in modules are defined to realize the content adaptation. The plug-in modules, also referred to as *intermediaries*, are small computational entities which modify or customize the input content. IBM also developed a transcoding proxy product called “WebSphere Transcoding Publisher” (WTP) [16], in which WBI is used as the plug-in component.

ICAP, i.e., the Internet Content Adaptation Protocol [12], is a protocol devoted to standardize client-server and object-based content adaptation for HTTP services. ICAP is simple in its design. In ICAP, clients can pass HTTP messages to ICAP the servers for transformation or other adaptation processes. The server then provides its transformation service on messages and sends the modified messages back to the clients, where the messages adapted can be either HTTP requests or HTTP responses.

In the iMASH framework [22], a modular and extensible Content Adaptation Pipeline (CAP) architecture is proposed. The middleware component within this architecture can perform content adaptation on any complex data types which are not limited to text and graphic images. This framework has three primary modules: data characterization function, adaptation command generator, and content adaptation executor. These three units form a pipeline to deal with the input data. The major contribution of this architecture is to decompose the prime actions for content adaptation into separated functionalities and well-defined components.

## 2.2 Comparison with Related Works

Almost all those frameworks, except iMASH [22], are mainly focused on the HTTP protocol. They basically function as an HTTP stream filter/rewriter to modify the content of the requested file. Due to the restriction of the HTTP protocol, the transcoding function of those frameworks will not be able to reuse the transcoding modules to transcode via other protocols. In contrast, the transcoding module of VTP aims at the content transcoding itself, and can be applicable to different protocols other than the HTTP. It is important to note that with the addition of a new service agent for the desired communication protocol, the original transcoding module can still remain intact.

As mentioned earlier, one major advantage of our VTP architecture is the usage of the controllable script language. Users or content providers may send “Transcoding Preference Script”

(TPS) to the proxy server, and the proxy server will perform appropriate transcoding procedure according to the instruction of TPS. The required transcoding agents and service agents will be automatically downloaded and installed from the specified Internet server. Those software agents will then be initiated in a Java-based execution environment to provide services. Such a code-on-demand characteristic further enhances the adaptability of VTP to various demands and reduces the memory usage of the proxy server.

### 2.3 Composite Capabilities / Preferences Profile (CC/PP)

Heterogeneous mobile devices usually differ in hardware, software, network connectivity, input/output, and browsing capabilities. The ability of transcoding the content according to the device capability is regarded as *context awareness*. The VTP architecture is designed to support context awareness. Each client has the option of registering the client capabilities on the proxy server so that the middleware can optimize the content transcoding operation. Context awareness can be achieved for each client to send a device profile describing the capabilities. Composite Capabilities / Preferences Profile (CC/PP) [24], developed by the W3C, is the prevalent standard of describing the device capability.

The specification of CC/PP follows the standard of Resource Description Framework (RDF)[2], which is based on XML and has a tree structure. The VTP architecture is designed to support the CC/PP<sup>2</sup> and to provide a parser for CC/PP in order to simplify the efforts of developing the transcoding agents. A profile may contain numbers of components and each component has several attributes. Figure 4 shows an example of mapping the profile into a two-level tree structure.

The reasons that we adopt CC/PP specification are as follows. Although RDF is a framework of describing the device capability, the vocabulary structure is not clearly defined in RDF. If we only use RDF, ambiguity may happen since the device capabilities are allowed to be described in different words. On the other hand, in the standard of CC/PP, the words of describing the device capability are unified. Therefore, the CC/PP specification is adopted in the VTP architecture to describe the device capability.

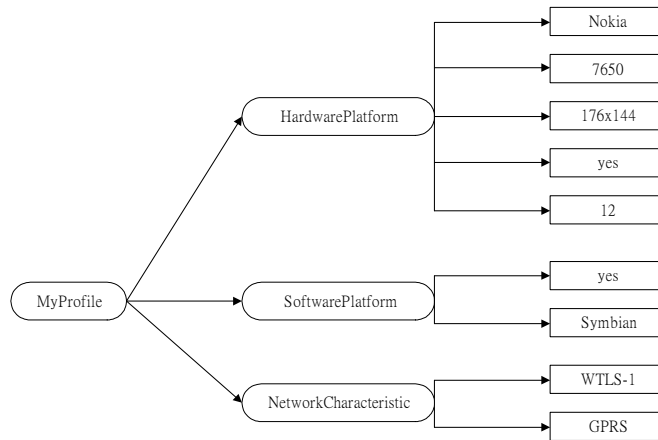


Figure 4: CC/PP profile in tree format

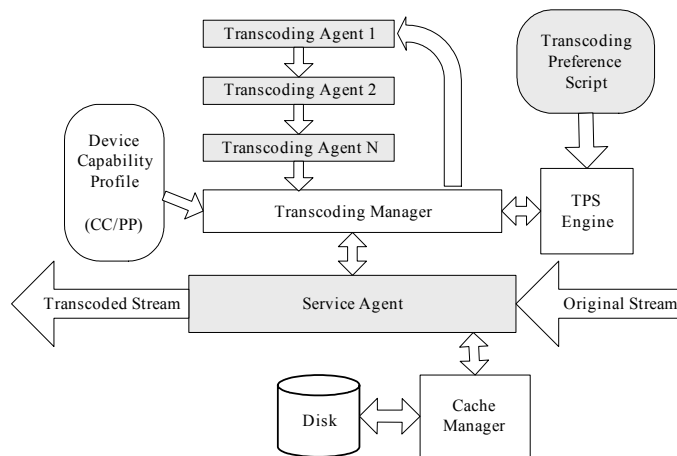


Figure 5: Overview of the VTP architecture

## 3 VTP ARCHITECTURE

### 3.1 Overview

To build a versatile transcoding proxy supporting heterogeneous data types and protocols, we design the transcoding proxy named VTP, standing for Versatile Transcoding Proxy, as a generic software agent container. When the transcoding proxy needs to process unknown data type or protocol, the proxy will automatically download the suitable software agent from Internet according to the instructions specified in the transcoding preference script. According to the difference in functionality, the software agents in VTP are classified into two main types, the transcoding agent and the service agent.

Figure 5 shows the detail of the VTP architecture. The transcoding agent in VTP is responsible for the data transformation. It is capable of transcoding the content from one MIME content type [14] to another MIME content type or to the same MIME type with degraded quality. The transcoding agent basically functions as a filter of the input data, and multiple transcoding agents are able to cooperate with each other to accomplish a transcoding task.

The transcoding agent in VTP can further be divided into two different types according to the awareness of the context. The first type is *context-aware* transcoding agent. This type of transcoding agent will perform transcoding based on the client device's CC/PP profile. In this type of transcoding agent, there is less but simpler control from transcoding preference script. The flexibility is high for the agent author but low for the user. The other type is dumb transcoding agent. This type of agent does not read the client device's CC/PP profile. It just transcodes the content by the transcoding hint provided by the transcoding preference script. This type of transcoding agent leaves most of the flexibility to the transcoding preference script provider.

The service agent in our framework is used to abstract the protocol details. With the service agent, we can easily apply the transcoding technology to many existing or future Internet application protocols. For example, to transcode the HTTP protocol, we shall instruct the proxy system to employ a "HTTP service agent". To transcode POP3 protocol, we instruct the proxy system to employ a "POP3 service agent". The user can dynamically instruct the proxy to download the required service agent, to extend the functionality of the proxy system, and to transcode the protocol that is understood by this service agent.

The transcoding preference script (denoted by TPS) in VTP is a scripting language based on the JavaScript to provide the mechanism which allows the user or the server to determine the agent

---

<sup>2</sup>For some special attributes, the description of the values is not clearly defined conscientiously in the CC/PP structure. For example, for a certain attribute, some developers may use the term "1", whereas others may use "True" to describe the value. To remedy this drawback, the standard of UAPProf [5] has been proposed above the CC/PP in WapForum. In UAPProf, the exact description of the attribute is defined to guarantee the vocabulary interoperability. The profiles adopted in this paper follow the CC/PP structure defined in UAPProf.

of processing the desired content, the time to perform transcoding, and also the location where the agent should be downloaded. In addition, a transcoding preference script can pass the *transcoding hints*, i.e., the parameters notified by TPS, to the software agent.

The VTP architecture is designed based on the agent system. Compared to the transcoding proxy in the prior research, the VTP architecture has the advantage of scalability and flexibility. Basically, VTP provides default software agents for basic requirements of the clients and the server. In order to satisfy the special requirements of transcoding, a specific transcoding agent can be either downloaded from the Internet or developed by clients/servers. Moreover, once a new protocol is established, we can create a new service agent and add it to VTP. Therefore, VTP can not only transcode via any kinds of existing protocols in application layer, including FTP, SMTP, POP, DNS, and HTTP but also deal with the specific protocols defined by the client or the server. The only restriction is that the VTP should be informed of the client-defined or the server-defined protocols via the available protocols in VTP.

## **3.2 Primary Components of VTP**

### **3.2.1 Transcoding Agents and Service Agents**

In the VTP architecture, an agent is a clip of small Java bytecodes which can be downloaded from the Internet to provide extension and enhance the functionality of the original transcoding proxy. A Java virtual machine provides a secure environment and isolation between different software agents. This is the reason of choosing Java as our implementation language.

The agent classes in the VTP framework contains BasicAgent, TranscodingAgent, and ServiceAgent. A new service agent can be created by simply inheriting the class of ServiceAgent and overwriting the necessary methods. Likewise, a new transcoding agent can inherit the class of TranscodingAgent as well. BasicAgent is the ancestor of TranscodingAgent and ServiceAgent. It defines the basic behavior of the software agents in VTP, and cannot be directly used. The definitions of those classes are shown in Figure 6.

When an agent is loaded from the Internet or the local disk, VTP uses the onBorn() method to activate the agent. The onService() or doTranscode() method will enable the agent to provide services or to perform transcoding. The onLeave() method is called when the agent is going to be unloaded from the memory. The getCacheCategory() method is used for the cache management.

### **3.2.2 Transcoding Preference Script (TPS)**

The transcoding preference script controls the behavior of the VTP server. Due to the flexibility and the simplicity, we adopt JavaScript as the language of the VTP architecture. Moreover, JavaScript also supports the control of VTP procedures, which make the VTP architecture more feasible to

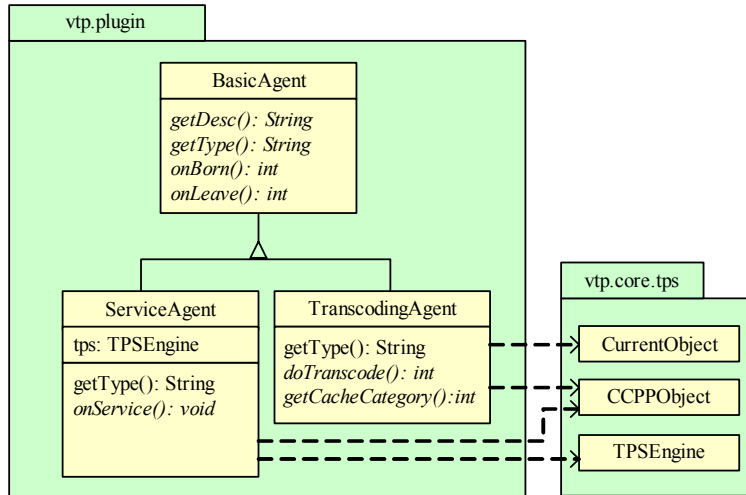


Figure 6: UML diagram for agent classes

Table 1: Available event handlers in TPS

Function	Description
onTranscode	called when a transcoding operation is going to be made
onLoad	called when the user makes a registration to the VTP server
onUnload	called when the user is no longer using the VTP proxy server
onConnect	called each time when the client connects to a service agent
onDisconnect	called each time when the client disconnects from a service agent

be implemented.

As shown in Table 1, the entry points in TPS provide pointers to some user specific processes in different internal states of proxy servers. TPS enables the VTP server to be programmable and provide flexible services. Users can load / unload agents by TPS and assign different agents to process the transcoding task in different contexts. TPS is able to read the attribute of CC/PP. We can thus use TPS to determine the property of a device, and ask the associated software agents to provide the suitable parameters for transcoding. CC/PP can be viewed as a vocabulary collection to describe the device capability without providing any transcoding function. Note that TPS can be developed by webpage designer, the device developer and the device user<sup>3</sup>. Therefore, the transcoding mechanism can satisfy the requirements of both the client and the server.

<sup>3</sup>For those users who are not interested in developing the script, there are two solutions listed as follows. The first one is that multiple agents for different purposes can be developed in advance by experienced programmer. These agents can be put in the website for users with different preferences to download. The other one is that an interface can be created so as to guide the users to develop the agents for their special requirements.

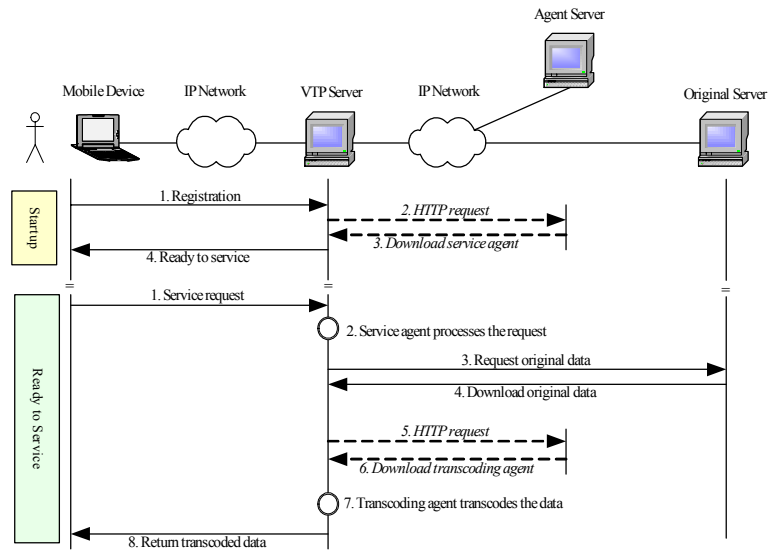


Figure 7: Scenario of VTP. (NOTE: Since VTP will cache the bytecode of the agent, the steps in italic font only need to be executed once for each agent.)

### 3.3 Execution Scenario of VTP

As shown in Figure 7, initially, a mobile device needs to register in the server. During the registration process, the device capability profile and the transcoding preference script are sent to the proxy server. The server will execute the transcoding preference script and download the designated service agent so as to set up the proxy server for the user. For example, the transcoding preference script in Figure 8 will set up the proxy for serving HTTP proxy and POP3 proxy function at ports 5000 and 5001, respectively.

Notice that the connection ports in VTP can be shared among different users. Hence there will be no port conflict in the `onLoad()` function made by different users. For example, John can use

```
function onLoad()
{
  transcoder.startService("sample.HTTPProxy","http://mmdb21.ee.ntu.edu.tw/http.jar",5000,null);
  transcoder.startService("sample.POP3Proxy","http://mmdb21.ee.ntu.edu.tw/pop3.jar",5001,null);
}
function onUnload()
{
  transcoder.stopService("sample.HTTPProxy");
  transcoder.stopService("sample.POP3Proxy");
}
```

Figure 8: Sample of configuring service agents

```

function onTranscode()
{
  if ( curobj.mimeType == "image/jpeg" && curobj.fileSize > 100000 )
    transcoder.execute("sample.JpegGray","http://mmdb21.ee.ntu.edu.tw/jpeggray.jar",null);
}

```

Figure 9: Sample of configuring transcoding agents

```

<HTML>
<HEAD>
  <!-- [TPS-begin]
    function onTranscode()
    {
      // Inclusion of Transcoding Preference
    }
  [TPS-end]-->
</HEAD>
<BODY>
...
  Web Content
...
</BODY>
</HTML>

```

Figure 10: Sample of embedded TPS in HTML

port 5000 for the POP3 proxy while Mary can use the same port 5000 for the HTTP proxy. The proxy system is able to identify the users by their source IP addresses of inbound connection and to dispatch the data stream to the corresponding service agent.

After the registration process, the proxy is ready for transcoding. Once a resource is going to be transcoded, the `onTranscode()` procedure is executed. For example, the transcoding preference script in Figure 9 will instruct the proxy server to transcode all JPEG graphics into the gray scale if their size is larger than 100KB. It is noted that in Figure 9, the transcoding agents can be initiated by the function `execute()` in the transcoder. The function `execute()` contains three parameters: the class name, the source address and the *transcoding hint* notified by TPS. The second parameter corresponds to the place where the agent is stored in the Internet. Therefore, as long as the agent is stored in a web server, the additional agents can be downloaded via the HTTP protocol.

Recall that TPS can be provided not only from the user side but also from the server side. A typical transferring mechanism of the TPS from the server side is to embed or hide the TPS in the data stream. The service agent is responsible for decoding the TPS from the data stream. For example, with the built-in HTTP service agent in VTP, it is easy to achieve server-directed Web page transcoding with the TPS embedded in HTML. The author of the Web page may put his/her

transcoding directives into the header section. Those directives correspond to the embedded TPS in a Web page, as illustrated in Figure 10. In addition to the default software agents and the agents configured by TPS, VTP can also download the software agents from the Internet. An additional server can be established to provide software agents for special requirements in the Internet. The situation of downloading agents from the Internet occurs when the VTP encounters unknown protocols or data types which the client or the server are not able to develop the transcoding agent by TPS. If the software is still unavailable from the Internet, the VTP will not be capable of performing the transcoding procedure. VTP will either keep all of the information or filter the object of unknown data type out.

Compared to other transcoding proxy systems, the VTP architecture suffers from the overhead of message passing. However, there are only two cases in which VTP performs worse. The first one lies in the registration process, in which the client has to send to the proxy server the device capability profile and the transcoding preference script. The other one occurs when the client or the server needs to deal with the unknown data types or protocols. In general, the message transmission of VTP is similar to that of the other transcoding proxy systems. On the other hand, since VTP use mobile agent technology to develop the transcoding proxy system, only part of the agents are activated for the specific clients/servers. Compared to the conventional transcoding proxy, which loads all transcoding functionalities into the system, VTP saves more resources such as memory and computing power.

### 3.4 Server Transcoding Preference

Traditional internet content adaptation systems tend to employ the transcoding operation based on some heuristics. The reason is that in those systems, the proxy usually does not have any knowledge about which data information is important. It is thus difficult for the proxy to determine the type of transformation or the degree of distillation to be applied to the content. However, with the VTP design, the server can pass “Server Transcoding Preference” (STP), including both the TPS and transcoding agents, to the proxy system. The preference specified will instruct the proxy system to take the suitable transcoding action on the content, and thus leads to a higher transcoding performance and quality.

Figure 11 shows a sample Web page which contains a large image. Suppose that the most important part of the image is the bird. Thus, it might be more preferable for the proxy to crop the image and to preserve the important part of the image, i.e., the bird, than to scale down the resolution of the whole image. For this purpose, the content creator in VTP is able to embed the script in Figure 12 into the header section of HTML using the technique presented in Figure 10.

The result is shown in Figure 13, where the left side of the figure indicates the situation when

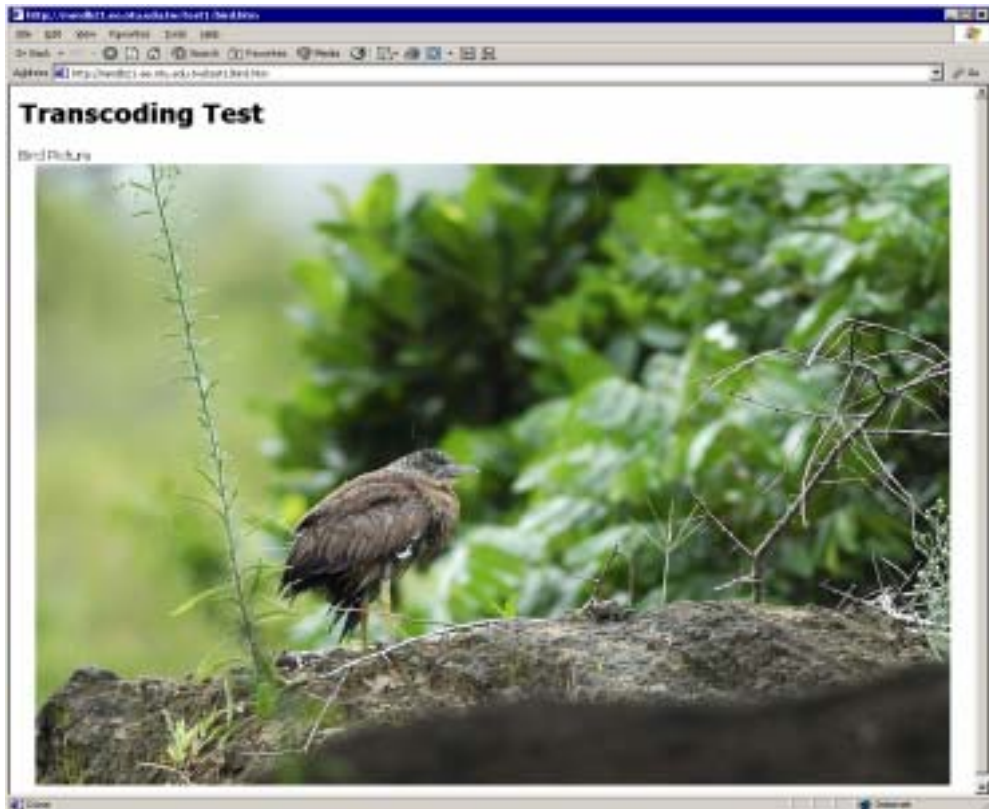


Figure 11: A test Web page for STP

```

<!-- [TPS-begin]
function onTranscode()
{
  var base = "http://mmdb21.ee.ntu.edu.tw/";
  var screen-Size = ccpp.getAttribute("ScreenSize");
  var ss = screenSize.split("x");
  if ( curobj.fileName.indexOf("bird.jpg") != -1 )
  {
    if ( ss[0] <= 240 )
    {
      var cropHint=transcoder.createHint();
      cropHint.put("left","200");
      cropHint.put("top","300");
      cropHint.put("width","240");
      cropHint.put("height","180");
      transcoder.execute("sample.CropImage",base + "cropimage.jar",cropHint);
    }
  }
}
[TPS-end] -->

```

Figure 12: TPS for cropping an image

STP is enabled and the size of the cropped image is 13041 bytes. In contrast, the right side of Figure 13 correspond to the situation without STP and the size of the downscaled image is 14918bytes.

From the result, it can be seen that with the help of proper STP, the most important information (the bird in this case) can be preserved in the transcoded version. The visual quality can also be significantly improved with a given file size. Such an advantageous feature also differentiate VTP from conventional transcoding proxy systems.

## 4 Scheme DCC-MPR

In this section, we describe the proposed scheme DCC-MPR to maintain the cache categories of a transcoding proxy dynamically. In Section 4.1, we first state mechanism DCC, standing for Dynamic Cache Categories, and the relevant procedures. Based on the *weighted transcoding graph* defined in DCC, algorithm MPR, standing for Maximum Profit Replacement, is proposed in Section 4.2 for a transcoding proxy to perform cache replacement. The symbols used in this section are given in Table 2. The database  $D$  is regarded as the collection which contains all possible objects and relevant versions. For each object  $i$ , the number of transcodable versions, i.e., the categories, is denoted by  $n_i$ . The original version of object  $i$  is denoted as  $o_{i,1}$ , whereas the least detailed version which cannot be transcoded any more is denoted as  $o_{i,n_i}$ .



Figure 13: Browsing with/without STP on Nokia 7650

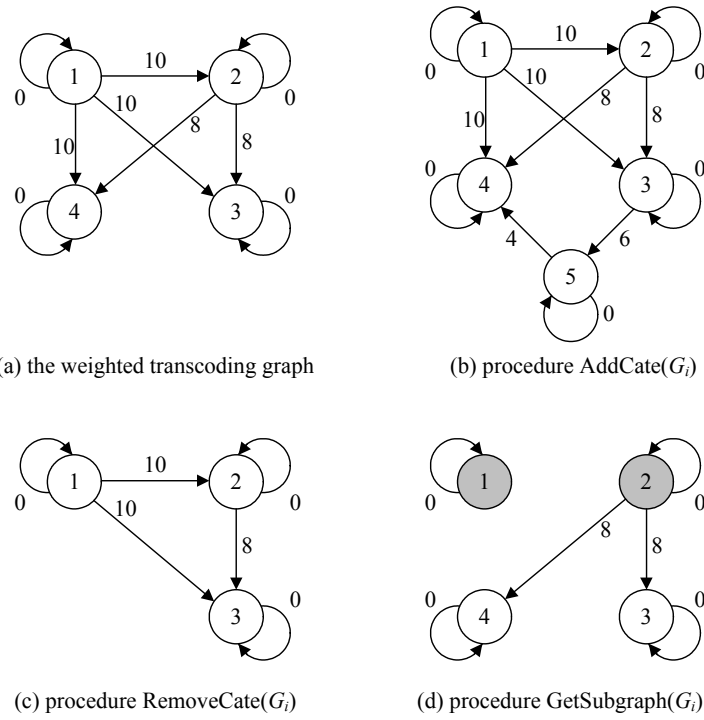


Figure 14: The illustrative example of mechanism DCC

Table 2: Description of the symbols

Symbol	Description
$D$	The database containing all requested objects
$n_i$	The number of transcodable versions of object $i$
$o_{i,j}$	Version $j$ of object $i$
$G_i$	The weighted transcoding graph of object $i$
$r_{i,j}$	The mean reference rate of $o_{i,j}$
$d_i$	The delay of fetching original object $i$ from the original server to the proxy
$s_{i,j}$	The size of $o_{i,j}$
$p_{i,j}$	The profit of $o_{i,j}$
$g_{i,j}$	The generalized profit of $o_{i,j}$
$PF(o_{i,j})$	The singular profit of caching $o_{i,j}$
$PF(o_{i,j1}, \dots, o_{i,jk})$	The aggregate profit of caching $o_{i,j1}, \dots, o_{i,jk}$
$PF(o_{i,j}   o_{i,j1}, \dots, o_{i,jk})$	The marginal profit of caching $o_{i,j}$ given that $o_{i,j1}, \dots, o_{i,jk}$ are already cached
$D_H$	The caching candidate set
$P_H$	The total profit of $D_H$
$S_H$	The total size of $D_H$

#### 4.1 Dynamic Cache Categories Mechanism

The proposed mechanism DCC contains several procedures to maintain the cache categories of the transcoding proxy dynamically. We first define the *weighted transcoding graph* to represent the relationship among different versions of an object. The weighted transcoding graph,  $G_i$ , is a directed graph with weight function  $\omega_i$ .  $G_i$  depicts the transcoding relationship among transcodable versions of object  $i$ . Each vertex  $v \in V[G_i]$  represents a transcodable version of object  $i$ . Version  $u$  of object  $i$ , i.e.,  $o_{i,u}$  is transcodable to version  $v$ , i.e.,  $o_{i,v}$  iff there is a directed edge  $(u, v) \in E[G_i]$ . The transcoding cost from version  $u$  to  $v$  is given by  $\omega_i(u, v)$ , i.e., the weight of the edge from  $u$  to  $v$ . Figure 14(a) illustrates the example of a weighted transcoding graph.

In conventional transcoding architecture such as TranSquid [19] and WTP [16], the number of transcoding categories of each object is predetermined. Therefore, the weighted transcoding graph of each object contains fixed vertices and invariant weight. However, in the VTP architecture, since the proxy is able to perform transcoding according to the user preferences, the categories of the cached object will change dynamically. In order to maintain dynamic cache categories, three procedures, i.e.,  $AddCate(G_i)$ ,  $RemoveCate(G_i)$ , and  $GetSubgraph(G_i, V')$  are defined. Procedure  $AddCate(G_i)$  will add a new vertex to  $G_i$  when a new version is created. It is noted that the transcoding relationship will be updated simultaneously once a vertex is added. Figure 14(b) shows

the result of procedure  $\text{AddCate}(G_i)$  after the vertex  $v_5$  is added to the original graph in Figure 14(a). On the other hand, procedure  $\text{RemoveCate}(G_i)$  is used to update the transcoding graph after a category is removed. Figure 14(c) depicts the result of procedure  $\text{RemoveCate}(G_i)$  after the vertex  $v_4$  is removed from the original graph in Figure 14(a). Procedure  $\text{GetSubgraph}(G_i, V')$  derives the subgraph which minimizes the transcoding delay when caching a certain set of versions of the object  $i$ . Note that  $V'$  denotes the set of the versions that the transcoding proxy tries to cache. Figure 14(d) illustrates the result of Procedure  $\text{GetSubgraph}(G_i, V')$ . Given  $V' = \{v_1, v_2\}$  and  $G_i$  in Figure 2(a), procedure  $\text{GetSubgraph}(G_i, V')$  will find the edge with minimum weight to reach  $v_3$  and  $v_4$ .

## 4.2 Maximum Profit Replacement Algorithm in Transcoding Proxy

According to the *weighted transcoding graph* defined in mechanism DCC, we design algorithm MPR, which achieves the maximum profit. A transcoding proxy should perform cache replacement according to the following concerns.

1. A version with high popularity is profitable to be cached.
2. A version with small item size is profitable to be cached.
3. A version which is helpful to reduce the transcoding delay of other versions is profitable to be cached.

Based on these principles, given the *weighted transcoding graph*  $G_i$ , the profit functions including the *singular profit*, *aggregate profit*, *marginal profit* and *generalized profit*, are defined as follows.

**Definition 1:**  $PF(o_{i,j})$  is defined as the *singular profit* of caching  $o_{i,j}$  while no other version of object  $i$  is cached.

$$PF(o_{i,j}) = \sum_{(j,x) \in E[G_i]} r_{i,x} * (d_i + \omega_i(1,x) - \omega_i(j,x)),$$

where  $E[G_i]$  represents the collection of all edges in graph  $G_i$ . It is noted that the reference rate  $r_{i,x}$  reflects the popularity, whereas the term  $(d_i + \omega_i(1,x) - \omega_i(j,x))$  is regarded as the *delay saving*.

**Definition 2:**  $PF(o_{i,j_1}, o_{i,j_2}, \dots, o_{i,j_k})$  is defined as the *aggregate profit* of caching  $o_{i,j_1}, o_{i,j_2}, \dots, o_{i,j_k}$  at the same time.

$$PF(o_{i,j_1}, o_{i,j_2}, \dots, o_{i,j_k}) = \sum_{v \in V[G'_i]} \sum_{(v,x) \in E[G'_i]} r_{i,x} * (d_i + \omega_i(1,x) - \omega_i(v,x)),$$

where  $G'_i$  is the subgraph derived from the procedure  $\text{GetSubgraph}(G_i, \{o_{i,j_1}, o_{i,j_2}, \dots, o_{i,j_k}\})$ .



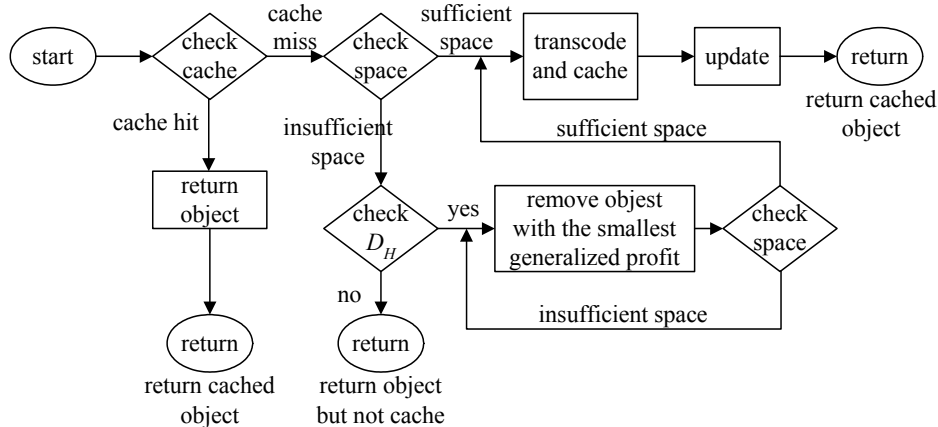


Figure 15: Flowchart of algorithm MPR

$$\leftarrow c[(i-1) * n + j - 1][z]$$

12. Determine  $D_H$  by tracing the elements in  $c[][]$

**end**

The complexity of procedure DP is  $O(|D| \times Z_c)$ , where  $Z_c$  is the cache size constraint. Therefore,  $D_H$  can be determined in polynomial time. Figure 15 depicts the flowchart of algorithm MPR. The detail of the algorithm is summarized as follows.

1. For each incoming query, check the cached items. If *cache hit* occurs, return the cached item.

2. If *cache miss* occurs, check the requested item size and the available space in the cache. If the available space of the cache is larger than or equal to the requested item size, and the requested version can be transcoded from some current version in the cache, perform transcode operation and put the transcoded version in the cache. If the available size of the cache is larger than or equal to the requested item size, but the requested object cannot be transcoded from any item in the cache, download the original version from the remote server, transcode it into the requested version, and put it in the cache. Finally, update the *profit* of the associated objects and  $D_H$ .

3. If the available space of the cache is smaller than the requested item size, check the *caching candidate set*  $D_H$ . If the requested item belongs to  $D_H$ , remove the item with the smallest *generalized profit* until the available space of the cache is larger than or equal to the size of the requested item. Put the requested item into the cache in the way the same as in 2, and update the *profit* of the associated objects and  $D_H$ .

Algorithm MPR can be divided into two phases. The first phase performs when the proxy has sufficient space to cache the requested item. In this phase, once an item is queried, it will be cached to increase the *profit* for future access. That is, the proxy will cache as many items as it can. The second phase performs when the proxy has insufficient space to cache the requested item. In the

Table 3: Data size of original and transcoded images with different profiles

CC/PP Profile	Screen Size	Colors	Image1 (1024x768)		Image2 (800x600)		Image3 (400x300)	
			Size	Ratio	Size	Ratio	Size	Ratio
Original		16777216	83,961	100%	52,785	100%	17,498	100%
Handheld PC	640x480	65536	15,009	17.9%	14,351	27.2%	6,915	39.5%
Color PDA	240x320	65536	3,652	4.3%	3,566	6.8%	3,433	19.6%
Nokia 7650	176x208	4096	2,502	3.0%	2,448	4.6%	2,375	13.6%

phase, the cache replacement is determined according to the priority of the requested item. The replacement policy of MPR is to keep the object with high priority and to remove the object with low priority. Since the *caching candidate set*  $D_H$  contains all the items with high priority, if the requested object belongs to  $D_H$ , replacement will occur to keep the requested data. We also use the *generalized profit*,  $g_{i,j}$ , to decide the order of removing items with low priority. Note that an item with low *generalized profit* tends to have large item size or low *profit*. Therefore, the item with the smallest *generalized profit* in the cache will be removed first. The removal procedure will continue until the proxy has enough space to cache the requested object with high priority.

## 5 PERFORMANCE MEASUREMENTS

In this section, we conduct several experiments to validate the performance of the VTP architecture. We measure the latency and power consumption of the VTP architecture in the first experiment. Note that in the second experiment, the email transcoding is implemented to verify the feasibility under via various protocols. In the Next two experiments, in addition to comparing the performance of the VTP architecture to other transcoding systems, we also inspect the performance of scheme DCC-MPR. Finally, we implement a webpage blocking service to validate the flexibility of the VTP architecture.

### 5.1 Transcoding Performance Measurement

The image transcoding is considered as one of the most important transformations for Internet content adaptation. In this experiment, we implemented an image distillation transcoding agent to execute the transcoding procedure. This agent will read the CC/PP profile of the client device, and check if the input image is larger than the screen of destination device. If yes, the transcoding agent will scale down the whole image to fit the screen size of the device. Independent of whether scaling down the image or not, this agent will try to further reduce the size of image by tuning the

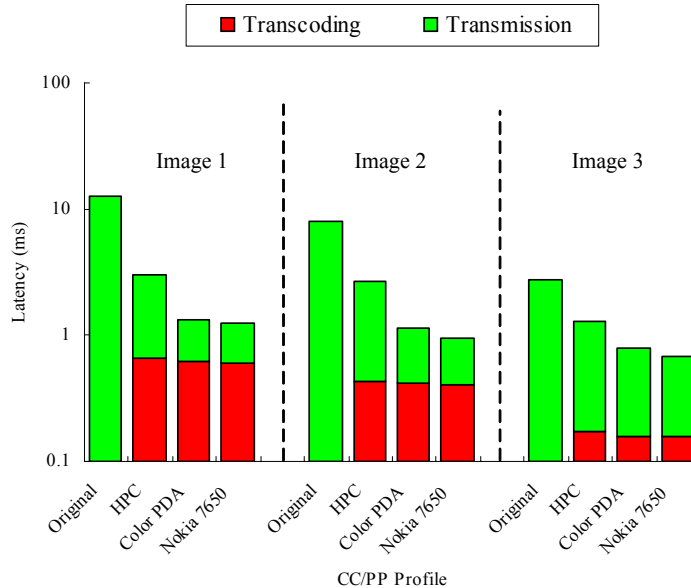


Figure 16: End-to-end latencies for images with different device profiles

JPEG quality factor. We have employed this transcoding agent for three different device profiles and test images (originally compressed with JPEG quality factor 85). The original and transcoded image sizes of those test images are shown in Table 3.

Figure 16 shows the client latency on a logarithmic scale for retrieving the original and the three transcoded versions of those test images. The latency is defined as the duration since the client issue a query until the required object is received. The bars are segmented to show the transcoding latency and transmission latency, separately. The transcoding process was taken on an AMD Athlon XP3000+ desktop PC running Microsoft Windows XP Professional with 512MB of memory. In this experiment, we can observe that the proposed VTP architecture can effectively reduce the transmission latency. The amount of the reduction of the transmission latency will be different for different mobile devices. e.g., the reduction of the transmission latency will be more significant for Nokia 7650 than for Handheld PC. The reason is that the transcoding agent of the VTP architecture will transcode the input image in such a way that the output image can fit the screen size of the mobile device. When the mobile device has larger screen size, the output image will be of larger size, which will cause the longer time to transmit the transcoded image. It is noted that in this experiment, the image was fetched via 56Kbps client-to-proxy bandwidth using a software traffic shaper. The traffic shaper is used to observe the worst-case performance of the experiment. For different clients, the bandwidth between the client and the proxy is different. We

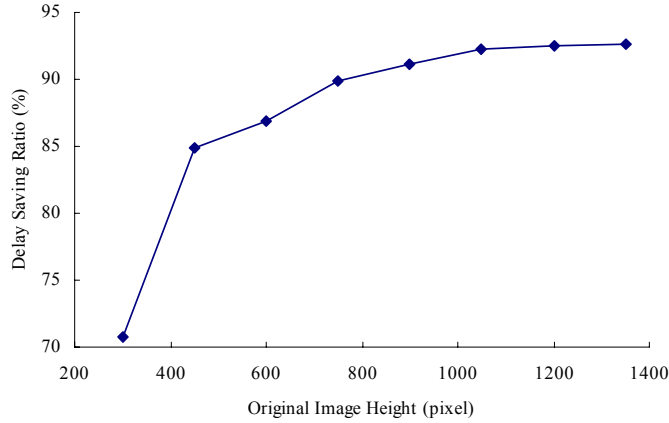


Figure 17: Delay saving ratio with different source image sizes (the image has the aspect ratio of 4:3)

regard 56kbps as the lower bound of the bandwidth. Since VTP performs well in lower bandwidth, it will be better in higher bandwidth. If the VTP architecture is used under the second-generation cellular system in which the bandwidth is lower than 56Kbps, the experimental results in Figure 16 will be more dominated by the transmission time. That is, the effect of transmission will be more significant than that of transcoding. Moreover, the reduction of the latency will be more distinguishable under the circumstances of lower bandwidth.

Though the transcoding process itself induces some extra time, the shorter latency can still be obtained due to the reduced data size. Note that the codes of the transcoding agent should to be downloaded from an Internet server when the agent is going to be used first time. Thus, an extra delay will occur. We also measured the cost of this extra delay, the download time of this image distillation agent is 0.02 seconds and takes another 0.17 seconds to do some initialization in our test environment.

Figure 17 further shows the *delay saving ratio* versus source images in different sizes. The formula we used to compute the delay saving ratio is:

$$R_{saving} = \frac{d - d'}{d} = 1 - \frac{d'}{d},$$

where  $d$  stands for the transmission time without transcoding, and  $d'$  represents the transmission time with transcoding. The result was measured using “Color PDA” profile with the same setup as before, which implies that the output image of the transcoding proxy will fit the screen size of the PDA regardless of the size of input image. We can observe that the performance of our image

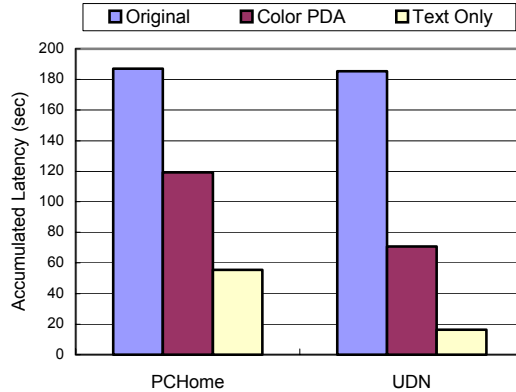


Figure 18: Time to retrieve ten emails with different device profiles

distillation transcoding agent will increase when the resolution of a source image grows. The reason is that the reduction of image size reflects the delay saving of the transmission. When the size of input image is getting larger, since the output image of the transcoding proxy is of the same size, the reduction of image size caused by the transcoding proxy will be larger. It is noted that the maximal value of the delay saving ratio is 100%. Therefore, the value of  $R_{saving}$  will be saturated when the size of the input image is much larger than that of the output image. In this experiment, the maximal delay saving ratio around 93% will be achieved.

## 5.2 Email Transcoding

In this experiment, we investigate the performance of the VTP architecture in email service. We adopt the same transcoding agent as in Section 4.1 for email distillation and transcoding. The service agent is designed for POP3 (Post Office Protocol) [21]. During the experiment, we measure the latency of downloading ten emails randomly chosen from PCHome member's mailing lists and UDN preferential epaper. The two CC/PP profiles used here are "Color PDA" and "Text Only", respectively. According to Figure 18, it takes around three minutes to download all untranscoded emails. For the users with the devices only supporting text-only file format, the VTP architecture chooses to filter all of the image contents out. As for the devices supporting color image displaying, architecture will performing transcoding mechanism to trade image fidelity for shorter latency. In addition to showing that the VTP architecture is suitable for various protocols in the application layer, this experiment also verifies the filtering process when there is no suitable transcoding mechanism for the client and the server.

Table 4: Results of energy saving tests

	VTP	No Proxy
Accomplished loops per battery	9680	965
Energy consumed per test loop	1.87 J	18.80 J
Battery life	329 min.	232 min.
Average power consumption	0.92 W	1.31 W

### 5.3 Energy Saving

We discuss in this experiment the issue of energy insufficiency, which is viewed as a fundamental problem for mobile devices. Since recent mobile devices have more built-in functionalities, the energy consumed is increasing. It is envisioned that the power problem will become more severe for years to come.

We use test loops to simulate a user downloading and viewing many pictures continuously on the PDA. Several procedures are contained in a test loop: 1. downloading the picture from an Internet server, 2. scaling the image resolution to fit the screen size, and 3. showing it on the screen. We have measured the number of test loops that can be accomplished with a full battery, and calculated the average energy consumed per test loop in Table 4. The test has been examined both with and without VTP for comparison. The proxy server hardware we used is the same as the first experiment in Section 4.1. The client device used to test is an iPAQ H3970 PDA running Windows CE 3.0 with Bluetooth as the network connectivity. The backlight of PDA is set to “always on” during the test.

From the statistics, we can see that in the VTP architecture, the total energy consumed per test can be reduced by 90%, which indicates a significant reduction on energy consumption. The benefit of energy reduction on a transcoding proxy is mainly caused by two reasons. First, we can obtain a shorter transmission time as we measured in the first experiment so that the energy used in wireless transmission can effectively be reduced. Second, with VTP we can shift some work (e.g., scaling down the image in this case) from the PDA side to the proxy side. If there are fewer tasks to be accomplished on the PDA, more energy saving can be achieved, which also indicates another advantageous feature of VTP.

### 5.4 Codesize and Memory Footprint

In our VTP framework, we divide a content adaptation system into several well-defined software modules. The required service agents and transcoding agents are downloaded from the Internet and then executed only when necessary. With this advantage of VTP, the code size and the

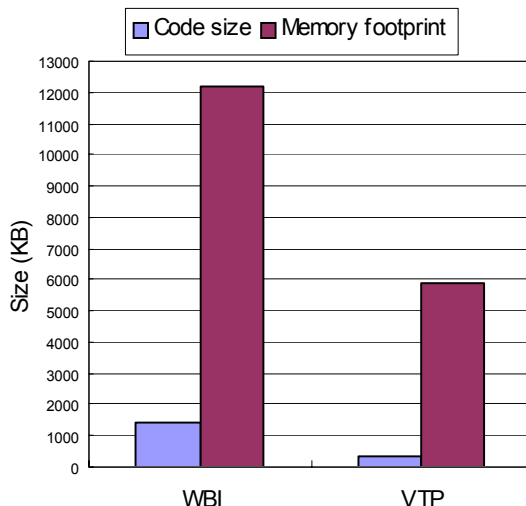


Figure 19: Comparison between WBI and VTP

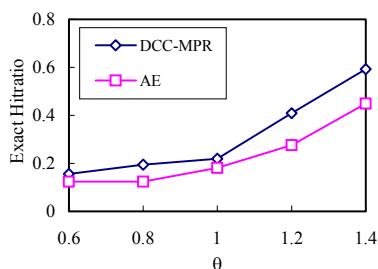
memory footprint of VTP are relatively small compared to other flexible intermediaries. Figure 19 compares the memory footprint and the codesize of the VTP architecture and WBI [16] developed by IBM WebSphere Transcoding Publisher. Note that the codesize represents the total codesize of both transcoding agents and service agents. From the experimental results, the codesize required by the VTP architecture is less than 20% of WBI. Likewise, the memory footprint of the VTP architecture is about half of WBI. The reason is that during the execution, only part of the agents in VTP are activated for the specific clients/servers. Compared to WBI which loads all transcoding functionalities into the system, VTP saves more resources. The result indicates that the VTP framework is compact in its size and suitable to run on a machine that is tight in resources such as memory and power.

### 5.5 Performances of Scheme DCC-MPR

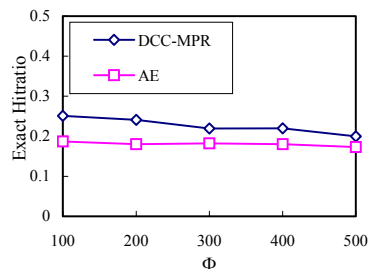
This experiment is designed to examine the performance of scheme DCC-MPR. The simulation model is designed to reflect the system environment of the transcoding proxy in Section 2. Table 5 summaries the definitions for some primary parameters. Without loss of generality, in the client model, mobile devices can be partitioned into five classes. That is, each data item can be transcoded to five different versions by the transcoding proxy to satisfy the users' requirements. The sizes of the five versions are set to be 100%, 80%, 60%, 40%, 20% of the original object size. A more detailed version can be transcoded to a less detailed one. Also, the distribution of these five mobile appliances is modeled as a device vector of  $\langle 15\%, 20\%, 30\%, 20\%, 15\% \rangle$ . As for the transcoding proxy model, we assume that there are 500 different objects, and choose the cache capacity to be  $0.1 * (\sum \text{object}$

Table 5: Simulation parameters

Parameters	Values
Number of objects	500
Skewness parameter ( $\theta$ )	0.6 ~ 1.4
Diversity parameter ( $\Phi$ )	100 ~ 500
Cache capacity	10%
Channel bandwidth	1 unit/sec
Transcoding rate	20 unit/sec



(a) when the skewness parameter is varied



(b) when the diversity parameter is varied

Figure 20: The exact cache hit ratio v.s. (a) the skewness parameter and (b) the diversity parameter

size). The reference rate of each object is generated by Zipf distribution  $r_i = (\frac{1}{i})^\theta \sum_{j=1}^N (\frac{1}{j})$ , where  $\theta$  is viewed as a *skewness parameter*. The size of each object is represented by the uniform distribution between  $(0, \Phi]$  units, where we define  $\Phi$  as the *diversity parameter*.

We compare the proposed DCC-MPR to the algorithm AE adopted in [10]. During the experiments, unlike the various metrics used in [10], we choose the tightest one, the *exact hit ratio*, to validate the performance of DCC-MPR. The *exact hit ratio* is defined as the fraction of requests which are satisfied by the exact versions of the objects cached. This metric is also motivated by the fact that we usually intend to provide an exact version to users (rather than an overqualified one) for effective bandwidth use. Figure 20 shows the *exact hit ratio* of DCC-MPR and AE as the value of  $\theta$  and  $\Phi$  varies. Since the comparison between AE and other schemes has been made in [10], we focus on comparing the performance between DCC-MPR and AE in this paper. In Figure 20(a), the *exact hit ratio* increases as the value of  $\theta$  increases. This phenomenon can be explained as follows. The *profit* is composed of two components: the reference rate and the delay saving. The increase of the skewness will make the *profit* dominated by the reference rate. Also, the increasing skewness enhances the locality of the object requests. Therefore, the *exact hit ratio* increases as the value of  $\theta$  increases. On the other hand, Figure 20(b) depicts that the *exact hit ratio* will slightly

decrease as the value of  $\Phi$  increases. Since the increase of  $\Phi$  will granulate the object size, it will be more likely that the cache space may not be utilized very thoroughly, which affects the performance of scheme DCC-MPR.

From the above two figures, we observe that the proposed DCC-MPR outperforms algorithm AE [10] prominently. The reason is as follows. Given the *profit*, object size and available cache space, the cache replacement in transcoding proxy can be modeled as a *0-1 knapsack problem*. In algorithm AE, a heap is built according to the value of the *generalized profit* of each object. If the proxy has insufficient space to cache all of the objects in the heap, the object with the smallest *generalized profit* is excluded. That is, algorithm AE always caches the objects with higher *generalized profit*. Such behavior in algorithm AE can be viewed as a *greedy algorithm*. Note that the *greedy algorithm* achieves optimal solution only in the *fractional knapsack problem* [13]. In solving the *0-1 knapsack problem*, since scheme DCC-MPR performs based on the *caching candidate set*, which is generated according to the concept of *dynamic programming*, DCC-MPR hence consistently outperforms AE.

## 5.6 System Extensibility

Owing to the programmable feature, the VTP has a wide range of flexibility and extensibility. In the final experiment, we have developed an adult Web page block service with VTP as an illustration. This service is composed by two transcoding agents which in essence inherit the TranscodingAgent class in Figure 6. One is SexHtmlBlocking transcoding agent, and the other is SexImageBlocking transcoding agent. While encountering any relevant keywords in the HTML, the SexHtmlBlocking transcoding agent will transform the original content into a paragraph of error message to inform the user this page has been blocked. In contrast, the SexImageBlocking transcoding agent works in the image domain. This agent analyzes the spectrum distribution of the image, and if finding a pornography, this agent will block the image. With the help of those two agents, the system is very efficient to prevent children from accessing sex and adult information. Figure 21 shows this adult Web page blocking service built by the VTP architecture. In Figure 21(a), if a legitimate web site is access via the transcoding proxy, the webpage can be displayed on the PDA of a mobile user. On the other hand, for the web site containing sex and adult information, the webpage will be blocked, as shown in Figure 21(b).

We comment that with its flexible framework, VTP can also be used in many other content filtering applications, such as virus scanning, language translation, and user activity logging. This explains the reason we call the transcoding proxy “versatile”.



(a) a legitimate web site



(b) a blocked web page

Figure 21: Display of the SexHTMLBlocking agent

## 6 Conclusion and Future Work

In this paper, the VTP architecture is proposed to provide a more flexible and extensible service for a transcoding proxy. Based on the agent system, the VTP architecture has several advantageous features such as low resource consumption and high flexibility. Moreover, in order to enhance the effectiveness of VTP, scheme DCC-MPR is also proposed to achieve dynamic cache categories and perform cache replacement. Experimental results have shown that the proposed VTP architecture and its corresponding scheme DCC-MPR have better performances in many aspects compared to the conventional transcoding proxy system.

We plan to extend this work in several directions. In the practical aspect, we intend to enhance the accessibility of the VTP architecture. In addition to dealing with more kinds of multimedia objects such as video and audio, we will also aim at tracking the access patterns of the user and providing different services for different users according their patterns. Moreover, we also plan to design an interface to facilitate the usage of the transcoding preference script. As for the theoretical aspect, since algorithm MPR is based on the concept of dynamic programming, it requires high complexity to determine the caching candidate set  $D_H$ . To reduce the workload of the transcoding proxy, it is possible to design an approximation algorithm to achieve high quality with much lower complexity. On the other hand, the design of a distributed architecture of VTP is also helpful to

balance the workload evenly across all transcoding proxies within the local cluster.

## References

- [1] ECMA, Standard ECMA-262 ECMAScript Language Specification. <http://www.ecma-international.org/publications/standards/ECMA-262.HTM>.
- [2] Resource Description Framework (RDF) Model and Syntax Specification. <http://www.w3.org/TR/REC-rdf-syntax/>.
- [3] R. Barrett and P. P. Maglio. Intermediaries: An approach to manipulating information streams. *IBM Systems Journal*, 38, 1999.
- [4] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. An Active Transcoding Proxy to Support Mobile Web Access. In *the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.
- [5] M. H. Butler and H. P. Laboratories. CC/PP and UAProf: Issues, Improvements and Future Directions. *Technical Report HPL-2002-35* <http://www.hpl.hp.com/personal/marbut/deliverycontextFinal.html>, 2002.
- [6] V. Cardellini, P.-S. Yu, and Y.-W. Huang. Collaborative Proxy System for Distributed Web Content Transcoding. In *ACM CIKM*, 2000.
- [7] S. Chandra, C. Ellis, and A. Vahdat. Differentiated Multimedia Web Services using Quality Aware Transcoding. In *IEEE INFOCOM*, 2000.
- [8] S. Chandra, A. Gehani, C. Ellis, and A. Vahdat. Transcoding Characteristics of Web Images. In *Multimedia Computing and Networking (MMCN-01)*, 2001.
- [9] C.-Y. Chang and M.-S. Chen. Exploring Aggregate Effect with Weighted Transcoding Graphs for Efficient Cache Replacement in Transcoding Proxies. In *IEEE ICDE*, 2002.
- [10] C.-Y. Chang and M.-S. Chen. On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies. *IEEE Transaction on Parallel and Distributed Systems*, 14(6), 2003.
- [11] C.-Y. Chang, M.-S. Chen, and B.-H. Huang. An H.323 Gatekeeper Prototype: Design, Implementation, and Performance. *IEEE Transactions on Multimedia*, 2004.
- [12] J. Elson and A. Cerpa. the Internet Content Adaptation Protocol. *IETF RFC 3507*.
- [13] T. C. et al. Introduction to Algorithms. *McGraw Hill*.

- [14] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions. *IETF RFC 2046*.
- [15] 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 Communication*, 5(6), 1998.
- [16] IBM. WebSphere Transcoding Publisher. <http://www-3.ibm.com/>.
- [17] B. Knutsson, H. Lu, and J. Mogul. Architecture and Pragmatics of Server-Directed Transcoding. In *the 7th International Workshop on Web Content Caching and Distribution*, 2002.
- [18] P. P. Maglio and R. Barrett. Intermediaries Personalize Information Streams. *Communications of the ACM*, Aug 2000.
- [19] A. Maheshwari, A. Sharma, A. Ramamritham, and K. Shenoy. TranSquid: Transcoding and Caching Proxy for Heterogenous E-Commerce Environments. *Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems*, 2002.
- [20] R. Mohan, J. R. Smith, and C. S. Li. Adapting Multimedia Internet Content for Universal Accesses. *IEEE Trans. on Multimedia*, 1(1), 1999.
- [21] J. Myers, C. Mellon, and M. Rose. Post Office Protocol. *IETF RFC 1939*.
- [22] T. Phan, G. Zorpas, and R. Bagrodia. An Extensible and Scalable Content Adaptation Pipeline Architecture to Support Heterogeneous Clients. In *IEEE ICDCS*, 2002.
- [23] M. Sniedovich. Dynamic Programming . *M. Dekker*, 1992.
- [24] W3C. Composite Capability/Preference Profiles (CC/PP) A user side framework for content negotiation. <http://www.w3.org/TR/NOTE-CCPP/>.
- [25] H.-L. Yang. Design and Implementation of an HTML-WML Translator. *Master Thesis Computer Science Department NTHU*, 1999.