An XML Based Solution to Delivering Adaptive Web Content for Mobile Clients

Wenzheng Gu and Abdelsalam (Sumi) Helal

Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611, USA
{wgu,helal}@cise.ufl.edu

Abstract

Due to mobile device proliferation, content providers can no longer deliver only one version of their content to the users, as they need to deliver an appropriate form of content depending on the capabilities of the viewing devices. Web authors either create multiple versions of the same content for each device, or depend on some intermediaries to do the transformation. In this paper, we propose two new approaches to the adaptive mobile content delivery based on the XML metadata. The first approach, called partiality adaptation, allows web authors to create one version of an XHTML page whose various parts could fit various devices. The second approach, called versioning negotiation, improves the efficiency of content negotiation by eliminating the resource description files. We further present our adaptation algorithms on both client and server side to reflect user intents. We also analyze the performance of our approaches based on three experiments conducted on different mobile devices.

1. Introduction

Providing a suitable content and presentation for different client devices in heterogeneous environments is becoming increasingly important today. There is already a plethora of exotic electronic devices such as pagers, PDAs, and color-display cellular phones, with no sign that the diversity of their characteristics will diminish anytime soon. As mobile devices come of age, one simple web page is no longer universally valid.

Objects on the web can be classified into two categories: an XHTML page, also known as a container page, and a content file including many groups like video, image, text, and audio. Significant research has been done to enable to speedy content delivery of the second group; for example, the Internet content transcoding and distilling for many specific file formats [14,24,25,26]. In [14], a framework is introduced to transcode contents among those groups. In regards to the container page object type, a widely adopted approach is to use XML to present the content and use XSL to describe the presentation. To adapt to different devices or context, XSLT is used to convert the source XML to the destination XML file. Our research is focused on both categories, because they are closely related to each other. Whether or not to download the objects of the second category depends on the links in the container pages. Efficiently removing some embedded object placeholders in the XHTML page can significantly reduce the total number of downloaded objects.

To better understand the client capabilities, network connectivity, or user preference, metadata or annotation is required and ranges from simple to complex descriptions. Many simple metadata are incorporated into HTTP headers, such as user agent information and page expiration time. On the other hand, there exists a very complex metadata like CC/PP specifying client-side profile, including every detail of hardware and software. Moreover, markup languages embed the metadata into documents -- the tags. In our research, we use tags to deliver the following information:

1) the appropriate size of XHTML page including the appropriate number of embedded objects, and
2) the available versions of each object.

Content adaptation and negotiation are two major approaches to delivering a suitable version of web objects to mobile devices. Content adaptation usually refers to those transcoding and distillation methods processed on a proxy near the mobile users based on their profiles and network QoS. Content negotiation, on the other hand, is a process between the client agent and the content server. The latter stores several variants of an object under different identifications (URL). After negotiation, an appropriate version is identified and delivered to the user.

We use the two aforementioned approaches combined to achieve efficient content delivery to mobile clients. Our research goals are to achieve the following:

- Bridge the gap between the different variants associated with the same object, namely one copy for

devices with different capabilities. For example, by merging both WAP and HTML pages to one, we save the time to generate different pages on the origin server, and we lower the burden on the proxy to keep track of different versions of the same content.

- Position web caching of objects in a better place for content negotiation and adaptation rather than delivering and transcoding cached copies directly. For instance, by sharing the same pages with different devices, the size of the user group will be increased, and therefore the cache hit rate would be higher, and the bandwidth usage and response time to user would be reduced.
- Make better negotiation and adaptation by requiring no client device information or intermediary. The adaptation is based on an appropriate version other than a large complete page. This reduces the proxy workload and saves the bandwidth between the proxy and origin server.

The rest of this paper is organized as follows. In section 2, we discuss related work. In section 3, we describe the design details of our adaptation algorithms. In section 4, we present preliminary experimental results. Conclusions are drawn in section 5.

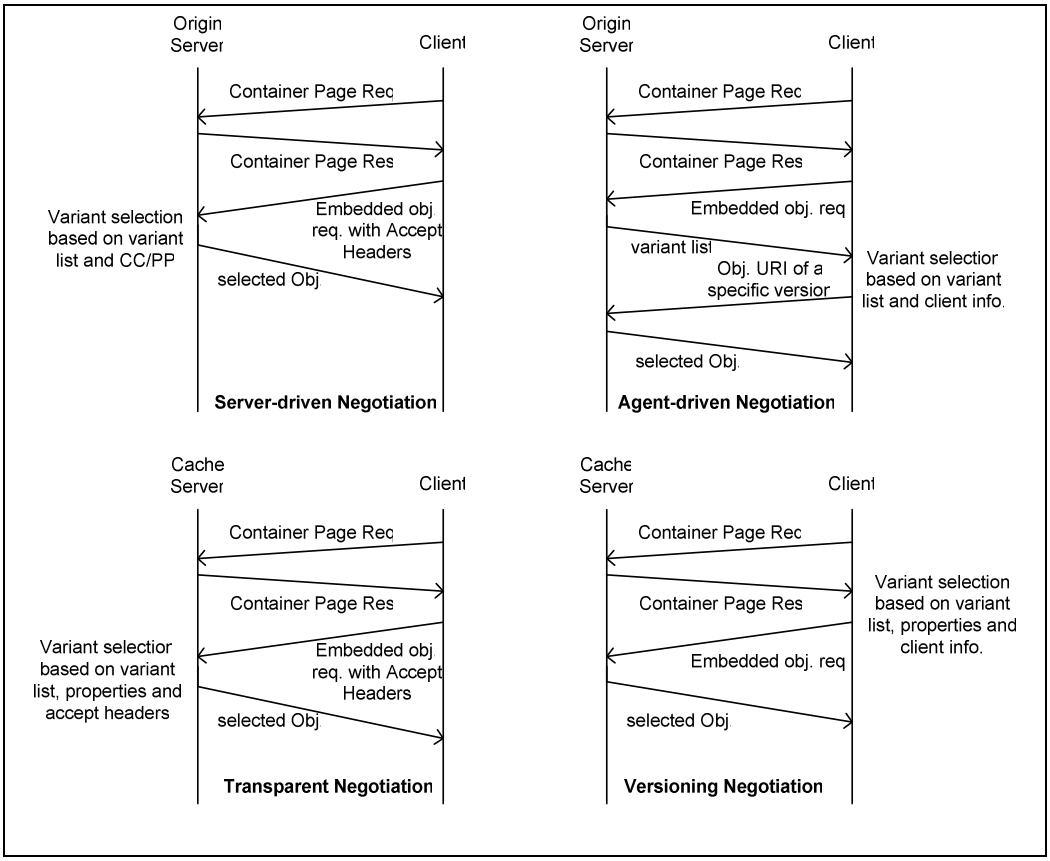


Figure 1. Negotiation Algorithms (Regarding the left two graphs, negotiation decision are made on servers. The right two graphs show that clients make the negotiation decision. But the upper right graph takes three round trips, whereas the bottom right one takes two. On the top two graphs, pages are returned from origin servers whereas at the bottom two, caching server can satisfy the request.)

2. Related work

There are many ways to describe device capabilities, such as HTTP Request Header Fields, CC/PP, WAP UAPROF, SyncML, Salutation, and UPnP [13,22]. All have advantages and disadvantages; for example, the HTTP request headers are too simple to describe the mobile device's capabilities and the user's preference, while CC/PP is too complex.

With the help of device description, an approach being widely adopted to solve heterogeneous device problems is content adaptation. Adaptation is most likely to be deployed on the Internet edge, like transcoding, translation, etc. The main drawback of this approach is that it breaks

various "end-to-end" HTTP properties which applications rely on. For example, many wireless gateways can transcode the original HTML page to a WML [9] page so it can be displayed on a WAP browser. But complicated web pages can make the heuristic algorithm on the intermediary difficult to apply. Also, the transformation may be against the authors' intention. There are many publications on content adaptation. Smith et al. [14] suggest manipulating web contents in two dimensions: fidelity and modality. Knutsson et al. [15] proposed an idea on server-directed transcoding which preserves HTTP end-to-end semantics. Chi et al. [16] gave a solution on image processing which can generate a higher resolution JPEG based on the previous low-resolution version. Shi et al. [8] presented a novel architecture to support caching of dynamic personalized content for mobile users. There are also other similar approaches including AvantGo, Web Clipping, ASP+.NET, PHP HawHaw Library, and XHTM/CSS[13].

Another approach is content negotiation. HTTP allows web site authors to put multiple versions (variants) of the same information under a single URL. To enable content negotiation, the variant list of the resource is usually kept as a file on the web server for later use. The list can also be retrieved from the file and sent out in the HTTP alternates headers. There exist three types of content negotiation mechanisms. They are server-driven

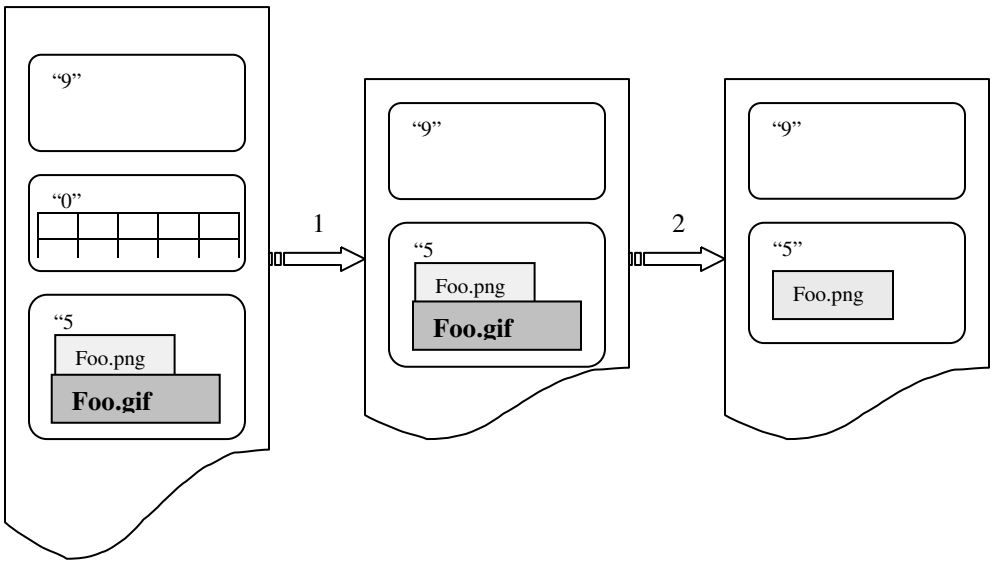


Figure 2. Processing by partiality adaptation and versioning negotiation. With priority tags, the number of objects to download is reduced in step 1 and with fidelity tags, the appropriate version is downloaded in step 2.

negotiation, agent-driven negotiation, and a combination known as transparent negotiation. None give satisfying solutions in terms of making decisions on the "best" copy. With server driven negotiation, it is hard to determine the "best" version because servers don't usually have complete user's information. The major drawback of agent-driven negotiation is that it introduces one more roundtrip time to fetch the variant list [1]. In RFC 2295 [11], the transparent negotiation is clearly defined and described. For the optimal transparent content negotiation, the variant file can be kept on a web caching server to help the intermediary make the final decision because it usually knows the user better than the origin server.

Although this transparent negotiation takes advantage of the cached variant list, variant properties and web contents, the final decision is still made by the proxy server. It is similar to letting a waiter decide the complete order for a customer in a restaurant, instead of the customer themselves. Figure 1 illustrates the above three scenarios plus the versioning negotiation. At last, HTTP Remote Variant Selection Algorithm (RVSA) is an important part of the transparent negotiation, which is used to determine the best variant [1].

To break a web page (a container page and its embedded links) into several fragments is not a brand new idea. It was first applied to those dynamic generated web objects. In [2,3], the static contents of a web page are cached separately from the dynamic parts. Chi et al. [4] proposed a XML based mechanism to validate different fragments of a web page in order to preserve data integrity.

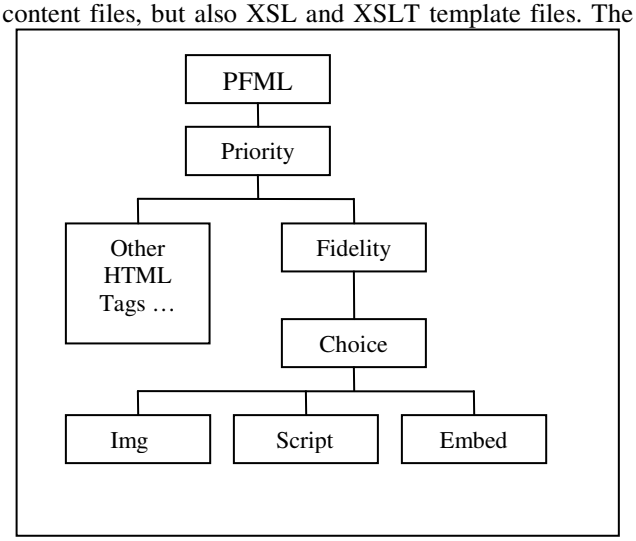
3. Design of partiality adaptation and versioning negotiation

3.1 Overview

Partiality adaptation and versioning negotiation are designed to deliver different web contents derived from the same page for heterogeneous networks and devices.

Briefly, there are two types of tags. The first is *priority tag*. Embedded Priority tags in an XHTML page indicate author's intention on how to partition a page into several fragments which are easier to be adapted. Later, a portion of the page can be downloaded upon the user's request. Furthermore, the priority tag can be used for not only content files, but also XSL and XSLT template files. The overall Internet traffic (including those for desktops) could be reduced by downloading the XHTML page in a smaller granularity. Moreover, the number of subsequent object requests could be reduced as well.

The second tag type is *fidelity tag,* which is used for versioning negotiation. Content negotiation requires a variant list for every object because it is the process of selecting the best representation for a given request when multiple representations are available. On the server side, content providers generate secondary versions of web pages that users can select from; for example, an HTML version or a WML version. A variant list is embedded in the HTTP header in the current negotiation mechanism.




```
<?xml version =”1.0”?>
<!DOCTYPE PFML SYSTEM “PFML.dtd”>
<!ELEMENT PFML (Priority*)>

<!ELEMENT Priority ANY>
<!ATTLIST Priority value (0|1|2|3|4|5|6|7|8|9) ‘0’>
<!ATTLIST Priority name CDATA #IMPLIED>

<!ELEMENT Fidelity (choice*)>
<!ELEMENT choice (img* | script* | embed*)>
<!ATTLIST choice sourceQuality CDATA ‘1’
                 type CDATA #IMPLIED
                 charset CDATA #IMPLIED
                 language CDATA #IMPLIED
                 feature CDATA #IMPLIED >
```

Figure 3. Hierarchy of PFML elements and PFML DTD

In our approach, fidelity tags convey the object variants message to the clients directly and precisely. This enables users to make their own decision in the first place. Different fidelity descriptions correspond to different attributes of content objects. For example, the language and charset of text, and the size and resolution of images. With the help of fidelity tags, a user agent can easily understand how many different presentations are associated with one embedded object and which objects are more suitable to display. Therefore, a clear decision can be made directly by the end user.

Priority tags not only maintain the end-to-end communication feature of HTTP, but also give both ends a better way to communicate. Publishers can use them to indicate their concerns on different objects in terms of priority. Clients can use them to fetch web objects as desired. And intermediaries can apply the content services accordingly.

Figure 2 shows how the content adaptation and negotiation work. In step 1, the origin page is tailored to the one without the lowest priority portion as the response to the user's first request. This adaptation work can be easily done on either the original site or an intermediary under the indication of priority tags. In step 2, the lower resolution image is selected according to the variant list embedded in the container page in the first response. The negotiation is much more efficient because the adaptation has generated a smaller page with fewer embedded objects. Therefore the mobile devices can afford to process it since fewer selection algorithms need to run.

3.2 Web caching in partiality adaptation and versioning negotiation

Our approach improves the performance of content negotiation by fully taking advantage of the current web caching architecture. In regards to web caching, server-driven adaptation is not always a good negotiation solution because it always bypasses the caching server. Regarding transparent negotiation, the optimized version requires not only the cached web objects, but also client profiles, object variant and property information. This would result in a

significant modification of the current web caching functionalities. Versioning negotiation gives a better solution. The requested XHTML pages are embedded with objects variant lists. As the bottom right part of Figure 1 shows, the best representation of an object can be automatically selected by the user agent. Thus an explicit URL request comes from a user agent as a normal request without negotiation meta-data, and all the cached web objects can be used as normal.

In the versioning negotiation, network traffic is reduced. First, the server need not fetch complicated user agent information, such as CC/PP. Second, compared with the agent-driven negotiation, versioning negotiation doesn't have one more round trip to send the variant list to the users.

```
<HTML>
<!--Foo's personal Web site. -->

<HEAD>
    < TITLE> Foo's Home </TITLE>
</HEAD>

<BODY>
    <!- - self-introduction- ->
    <P> I am … </P>

    <!- -Personal picture - ->
    <IMG SRC="Foo.gif" BORDER…>

    <!- - My interests - ->
    <P> I like sports and music… </P>

    <!- -friends' link - ->
   <P>Foo1 < A HREF = HTTP://…></P>
   <P>Foo2 < A HREF = HTTP://…></P>

    <!- - contact information - ->
    <P> Phone #:… </P>
</BODY>

</HTML>
```

```
<?xml version = "1.0"?>
<PFML>
<Priority name ='foo_1' value='9'>
<!--Foo's personal Web site. -->
  <HEAD>
     <TITLE> Foo's Home </TITLE>
  </HEAD>
</Priority>
  <BODY>
   <Priority name ='foo_2' value='9'>
   <!- - self-introduction- ->
   <P> I am … </P>
   </Priority>
     <Priority name ='foo_3' value='5'>
      <!- -Personal picture - ->
          <IMG SRC="Foo.gif" BORDER…>
      <!- - My interests - ->
          <P> I like sports and music… </P>
      <!- -friends' link - ->
          <P>Foo1 < A HREF = HTTP://…></P>
          <P>Foo2 < A HREF = HTTP://…></P>
     </Priority>
     <Priority name ='foo_4' value='5'>
      <!- - contact information - ->
          <P> Phone #: (123)456-7890 </P>
     </Priority>
   </BODY>
</PFML>
```

Figure 4. Foo's homepage implemented with HTML(left) and PFML(right)

smaller device, and send it to the user directly. This can shorten the response time. The remaining objects could be downloaded simultaneously and integrated with the previous part at last. Therefore, letting the mobile device users and traditional desktop users share the same web page potentially increases the chance of web cache hit rate, reduces Internet traffic, and lowers the response time to clients.

Our content adaptation approach also helps web caching. Priority tags allow different devices share the same copy of an XHTML file which is usually stored in the caching server. A mobile device can take advantage of the copy of a web page previously downloaded by some other device, for example a desktop, in a caching hierarchy. Instead of going all the way to the far end of the Internet, the client with less capabilities can obtain a subset of the page by extracting the content marked with higher priorities. On the other hand, a device with more capabilities can use the partial copy of a web page downloaded previously by a

3.3 Tags in partiality adaptation and versioning negotiation

There are two types of tags and associated attributes in our approach. These tags can be incorporated into other XHTML languages so more complicated functionalities can be implemented while the aforementioned adaptation and negotiation goals can still be achieved. Figure 3

illustrates the overall structure of an XHTML page with priority and fidelity tags and its Document Type Definition (DTD). Here we name the language with the new tags as *PFML* for documentation purpose.

3.3.1 Priority Tags. Priority tag is used to divide a web

page into several portions in order to cater to the devices with different capabilities. Each part is assigned a priority value. The author can use the attribute values of Priority tag to express their intentions. Value '9' indicates that the corresponding content has the highest priority which must be sent out when its URL is requested, whereas the value '0' means the corresponding content is the least important, compared to other parts of the web page. By default, the

```
<Fidelity>
  <choice sourceQuality= "1" type="img/gif">
      <img src="/images/foo.gif" width="276"
           height="110" />
  </choice>
  <choice sourceQuality="0.6" type="img/png">
     <img src="/images/foo.png" width="76"
          height= "30" />
  </choice>
  <choice>
      foo
  </choice>
</Fidelity>
<Fidelity>
  <choice sourceQuality= "0.9" type= "text/html"
          language= "en">
     <doc src="/document/paper.html.en" />
  </choice>
  <choice sourceQuality= "0.7" type="text/html"
          language="fr" >
     <doc src="/document/paper.html.fr" />
  </choice>
  <choice sourceQuality= "1.0" type=
          "application/postscript" language= "en" >
     <doc src="/document/paper.ps.en" />
  </choice>
</Fidelity>
```

Figure 5. The examples of using Fidelity tags. (The above examples are adopted from RFC2296 [1] with some modification.)

incorporated by applying Fidelity tags to them. Each pair of Fidelity tags quotes a list of variants for one URL. The choice tag specifies one of the versions of an object and its properties. Figure 5 shows two objects where there are three variants associated with each of them. The first object is an image file associated with three presentations. The image in GIF format has the best quality and biggest value is set to '9'. The values could be incremented or decremented automatically with the algorithms described in section 3.4.

An example is given in Figure 4. Mr. Foo has a personal web site. Whenever a request to his homepage is made, he always wants his self-introduction and contact information to be downloaded. So he assigns them priority '9'. And he defines the remaining part as optional with priority value set as '5'.

3.3.2 Fidelity Tags. Fidelity Tags are mainly used for content negotiation. Other than keeping a variant list in a file, web server can insert the lists and properties into an XHTML page where the corresponding web object is embedded. The lists and properties can be easily size. (The quality values are evaluated by web author and we adopt them from [1].) It is suitable for desktops and laptops. To target PDAs, pictures in PNG format are the best choice, because they have a little lower quality and relatively smaller size. For those pager and cellular phone users, a plain text can be displayed instead of the image. In the second example, the document in English and kept as a postscript file is the best version. English version in plain text is worse. And the French plain text is the worst copy in terms of quality. The variant selection can be done ideally on the user agent, which has the complete knowledge of the user's hardware and software platforms.

3.4 Automated Assignment algorithm of fidelity and priority values

3.4.1 Priority Assignment Algorithm

3.4.1.1 Server Side. An XHTML page first needs to be divided into several segments because the smaller the granularity, the easier to adapt to different devices. Each segment can be as big as a whole page, or as small as a table cell, a link to an object or even a word. Decision on how to fragment a page is made by either the page author or some automated programs. Regarding small web sites like personal sites mentioned in Figure 4, the author could fragment a page into several portions according to the semantic. Fragmenting big commercial sites like yahoo.com may be not an easy job for a human being, whose size is about 50k. Instead, an automated program can parse and fragment it. Some HTML tags could be considered as the delimiters, such as pairs of <h1> </h1> for headings,   for paragraphs, <frameset> </frameset> for frames, <table> </table>,<tr> </tr>, <td> </td> for tables and etc. Tools like Xpath could be used to locate these tags, then insert them in between the pair of <priority> </priority>. Certainly, an unique ID like foo_x in Figure 4 should be assigned to the segment for later use.

For each segment, the initial priority value could be assigned in two ways, which are similar to the above

fragment methods. If the authors fragment the page manually, they could also assign the priority values by themselves. In this way, they can show their intentions and have some controls on their pages. Otherwise, an automated program could traverse the whole page and assign value "9" to all the segments as a default value.

There are ten priority values range from 0 to 9 and only significant within the same page. That means the segment priorities from different web pages are not comparable. For each page, value 9 indicates the highest priority whereas 0 means the lowest. Having been assigned the initial value, the priority will be changed periodically based on the number of accesses to a segment collected from client agents. Figure 6 shows the algorithm. Basically, if a client agent makes the request to the same page, which it requested before, the number of clicks for each segment will be piggy-backed. And the total corresponding numbers would be updated accordingly on the server. The priority values will be recalculated periodically based on the number of clicks for each segment.

```
# Nc: total number of clicks; increment upon each click
# Ns: total number of segments of a page
#  t: a function to calculate a specific threshold with parameters
# Pi:  priority value for segment i
# Ti: the time stamp to generate the Pi
# Tnow: the current time
# Ci: total number of clicks on segment i
# Ci': total number of clicks on segment i sent from client agent

# executed on each access
for each segment i
{
        Ci ← Ci + Ci';
        Nc ← Nc + Ci';
}

# executed periodically
for each segment i
{
    # priority value increment
        if ( Ci  > t (Ci,Pi,Nc,Ns)   and   Pi < 9  )
    {
                Pi ← Pi + 1;
                Ti ← Tnow;
     }
    # priority value decrement
    # expired means the segment hasn not been touched for a
period
        else if ( Ti expired    and   Pi > 0 )
     {
                Pi ← Pi –1;
                Ti ← Tnow;
      }
}
```

Figure 6. Page Segment Priority Value Decision Algorithm

Moreover, along with each request, there is a request priority value sent by each client agent. It is used to reflect user's preference. Namely, what percent of the whole page would be interesting to download. If the value is 0, the complete original page would be sent out. If the value is 9, only the smallest portion with priority value 9 would be transferred. Generally, if the value is r, where 0<= r <=9, all the segments with priority value p (p>=r) would be sent out.

```
# Nj,c: total number of clicks on a page; increment upon each click
# Nj,s: total number of segments of a page
# Np: total number of pages
#  t:  a function to calculate a specific threshold with parameters
# Pj,i:  priority value for segment i
# Pj,c: priority value for a page
# Tj,c: the time stamp to generate the Pj,c
# Vk: the total number of pages having priority k, where 0<=k<=9
# Pa:  priority value for a client agent
# Ta: the time stamp to generate Pa
# Cj,i: total number of clicks on segment i, page j.

# upon each click
if (new page)
    Np ← Np +1;
    Initialize new (Cj,i)s to 0;
    Initialize new Pj,c to 0;
Cj,i ← Cj,i + 1;
Nj,c ← Nj,c + 1;

# at the idle time
for each page j
        Pj,c' ← Pj,c;
            # change priority of page
            for each segment i
                    if ( Cj,i > t(Nj,c,Nj,s)  and Pj,c > Pj,i)
                            Pj,c ← Pj,i;
                            Tj,c ← Tnow;
                    else if ( Tj,c expired)
                            if ( Pj,c < 9 )
                                    Pj,c ← Pj,c + 1;
                                    Tj,c ← Tnow;
            # change priority of agent
            if ( Pj,c <> Pj,c')
                k ← Pj,c';
                Vk ← Vk -1;
                    k ← Pj,c;
                    Vk  ← Vk +1;
                    if (Vk > t(Np)  and   Pa > k)
                            Pa ← k;
                            Ta ← Tnow;
            else if (Ta expires and Pa < 9)
                    Pa ← Pa + 1;
                    Ta ← Tnow;
```

Figure 7. Client Agent Priority Value Decision Algorithm

3.4.1.2 Client side. For a downloaded web page, client agent is responsible for counting the number of accesses to each segment. Namely, the agent maintains a tuple <URL/Segement_name, counts> for each segment in a page. Based on this data, the client agent changes its request priority value, which is set to 0 at the installation time. The algorithm, which is illustrated in Figure 7, could be running at the agent idle time without interfering with user's browsing. The request priority value of a client

agent is determined by the user's behavior. First, the priority value of a web page visited by the agent is calculated. This is based on the statistic values of segment clicks. Then by collecting the page priority values of all the visited pages, the priority value of the agent can be determined. Generally, two rules are applied for both page and agent priority decisions:

1. If the number of accesses to a segment/page is greater than a threshold, the current value is set to the minimum of current priority value and the segment/page value;
2. If a priority value hasn't been changed for a long time, the priority value is incremented.

3.4.2 Fidelity selection algorithm. As illustrated in Figure 5, the Remote Variant Selection Algorithm (RVSA) [1] could be used as the automated fidelity Selection Algorithm. In fact, this algorithm is more suitable for our versioning negotiation because the meta data in RVSA are used to describe web objects, but not the device information. Thus, with the description and the on hand device information, client agent could make a good choice. If RVSA is run on any server, the lack of device meta-data can potentially result in a biased decision

4. Experimental Evaluation

4.1 Experiments on Priority Tags

We conducted a series of experiments on the client and proxy sides. On the client side, three types of wireless devices were used. The first is a J2ME-enabled Motorola i85 cellular phone. It uses Nextel service which provides a maximum 19.2 kbps transfer rate. We consider this as a low-end device in terms of network performance, CPU power, and display size. The second one is an IBM 390 Thinkpad which uses a 802.11b compatible wireless card to connect to the access point in our lab. The connection rate was at 11 Mbps. This is considered to be a high-end device. The third one is an iPAQ 3800 PDA and uses the same network adaptor and network connection as the laptop but with less computation power and memory. We've also programmed a proxy server and let it run on a Sun Ultra 60 workstation. The proxy code includes several modules as a normal proxy server does. They are a server side module, responsible for setting up a connection with the web server; a client side module, in charge of the communication with clients; a cache management module; and a PFML parser. The two web servers we used were google.com and cnn.com. The HTML page of cnn.com is about 50k and frequently updated. On the other hand, the HTML page of google.com is less than 3k and rarely changed.

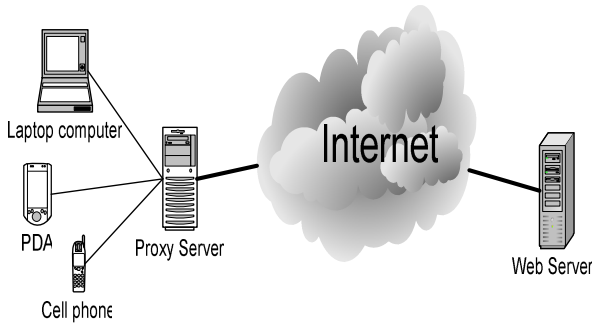


Figure 8. Experimental Environment

We designed three cases, as Figure 8 shows, to download a portion of the web page to the client which is about 1k size.

- Remote Case: the page is downloaded from the origin site. The client sends out a request to our proxy server, then the proxy relays the request to the origin site. Having received the whole page from the web, the proxy extracts the first 1k data and forwards it to the client.
- Extracted Case: we put the pages of the web sites onto the proxy server's local disk, and inserted some pairs of <Priority …> </Priority> tags into the origin pages. Upon the user's request, the parts marked with <Priority value= "1"> are extracted and sent back to the client.
- Cached Case: we put an extracted copy of the web site on the proxy, which is about 1k. When the user's request came, the copy was sent out immediately.

Figure 9 shows the total time measured between the user's sending out the request and receiving the desired page. The performances of cached and extracted cases are very similar, whereas the remote case has two or three orders of magnitude of larger retrieving time. Each node represents the average time collected from 49 runs (7 different times in a day and Sunday to Saturday in each

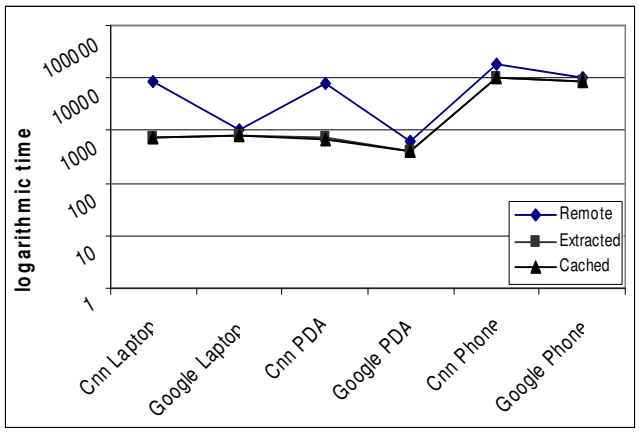


Figure 9 Comparison of total time to download pages to different devices

week). According to the experimental result, in regards to the cnn page, the average time to process a cache hit is about 3ms, to fragment a 50k cnn.com home page is about 15ms, and to download it from the web is approximately 5000ms. For a smaller page like Google's homepage, which is about 3k. the three values are 3ms, 4ms, and 500ms. The 500ms is due to its relatively long expiration time which results from pages downloaded from nearby proxy servers. The first observation is that to fragment a page on the local cache server is much faster than retrieving it. The second observation is how slow cellular phone-proxy-connection time offsets the benefit brought by the web caching in both the "CNN phone" and "Google phone" cases. On the other hand, when the wireless link transfer rate is at least one order of magnitude lower than that of the wired line and the size of the Web object is small, for instance 3k, the saved time is less significant compared to the delivering time on the current telecommunication network. But according to the experiment, it still saved about 1.5 seconds out of 10 seconds total delivering time. Therefore, to draw a more accurate conclusion on the performance of a priority tag on the slow wireless networks, we measured the delivering time on the cellular phone by sending out requests to 100 popular Web sites.

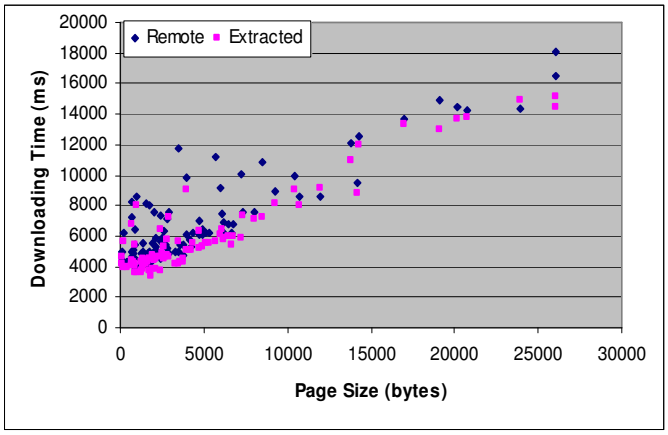


Figure 5-11 Simulation data on 100 Web sites retrieved by cellular phone.

The sites were collected from www.hot100.com with more than 10 categories. The sites are generated using Overture's search rankings for the query contained in the category. The 10 categories that we chose are listed in Table 5-2.

We used both Remote and Extracted Cases. With remote simulation, the pages were downloaded to the cell phone from the origin site. The downloading time is shown by the blue square in Figure 5-11. But this time, in our client code, the intermediary copies are allowed to fetch. Since the selected Web sites are very popular, pages are most likely to be downloaded from a proxy server close to client agents. Therefore, this is the best scenario for the Remote case, namely, users can always download a page very fast. In comparison, we stored pages on our Unix machine, and when users' requests come in, the pages are simply read through once and sent back to the phone directly. We did not extract any contents because we want to deliver the same sizes of the pages, otherwise the saved transferring time on a wireless network will dominate the overall amount of saved time. The Extracted Case measurement is shown with pink diamond dots in Figure 5-11. The average size of downloaded page is 4843 bytes. The average downloading time is 6875ms in remote simulation, whereas it is 5910ms in extracted simulation. Therefore, by allowing users to share the cached objects among heterogeneous devices with Priority tags, we are able to save 965ms to download an average-sized Web page. The performance is improved at least 14%.

There are several things which need to be taken into account. First, the workload on the proxy is pretty low, which may lead to less processing time of page extraction. Second, the proxy server is supposed to be put at the spot where the backbone of the Internet and the cellular network are connected. But in doing our simulation, it was not deployed this way. Most programs were coded in JAVA language except the one on PDA, which is in C#. But in [23], C# and Java are proven to be very close in TCP socket performance.

4.2 Experiment on Fidelity Tags

We conducted two simulations on both content negotiation with CC/PP and our Versioning Negotiation. We showed the experimental result that Versioning Negotiation outperformed the content negotiation with CC/PP. The simulation environment was set up as Figure 16 shows. The Web server is Apache. It hosts more than 1000 personal Web sites of CISE Department students and professors at the University of Florida so we are able to test the simulation with some real workload. Two CGI modules were put onto the server programmed with Perl. One is for CC/PP, the other is for Versioning Negotiation. The CC/PP module receives the users' requests, retrieves the device information from a third-party site, matches different versions with devices descriptions, and sends back the selected version to the client. We call our Versioning Negotiation module PAVN. It behaves like a normal Web server which simply returns pages. The matching process between devices and objects happens on the client side. We used a Motorola i95 cellular phone as a client and programmed with J2ME and MIDP 1.0. Elapsed

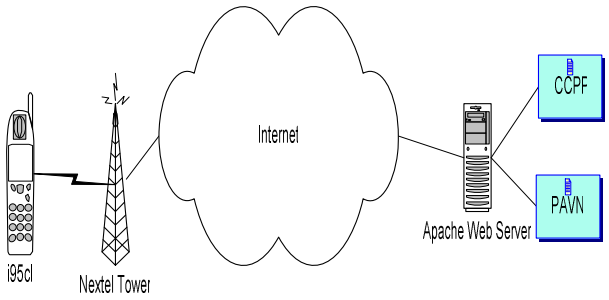


Figure 16. Simulation environment to compare performance on CC/PP and PAVN negotiation modules

time is measured on the cellular phone and displayed on the phone screen. The downloaded pages and objects are not displayed because for both methods, they take the same amount of time to render. The phone has 1.5M RAM, 160*120 screen, 8 bits color depth [31]. The CPU speed is not disclosed by Motorola, but it is estimated to be around 100MHz [32].

The simulation results are illustrated as a bar graph in Figure 17. The vertical axis indicates the round-trip time to fetch the Web objects. The horizontal axis shows the sizes of the downloaded objects. The plus sign on the object size means two objects were downloaded; for example, 1k+256 indicates that two objects were downloaded. It means the size of the index page is 1k, and the size of the embedded object is 256 bytes. Each bar shows the average time of 15 runs.

The Versioning Negotiation outperformed CC/PP in all 9 cases where both the CC/PP and PAVN algorithm are executed. The average saved time is about 870 milliseconds. To further explore the amount of the saved time, we analyzed the performance on the two server side modules carefully.

The two CGI modules are very short. The PAVN CGI simply gets the request, opens the corresponding file, and sends it back. The CCPP CGI uses the Library for WWW in Perl (LWP) to fetch the device CC/PP file, then gets the image format information, for example, the width, height and the resolution, and finally opens the requested file and returns it.

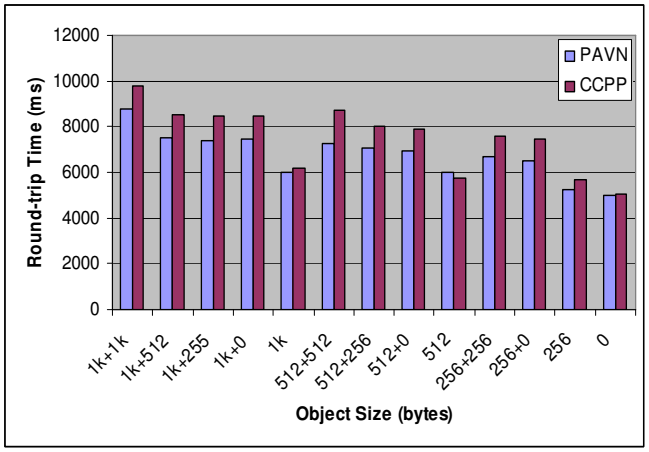


Figure 17. Simulation results on CC/PP and Versioning Negotiation

We used the UNIX *time* command as a benchmark to measure the two programmed modules. Figure 18 shows the details returned by the time command. For the CCPP module, the majority of the extra time was spent on retrieving the CC/PP file from the Internet. The user process consumes 0.78 second, the system calls on network takes 0.11 second, including file opening and network connection setting up, and the total elapsed time is 1.35 seconds. Therefore, it takes about a half second to fetch the CC/PP file from the Internet by subtracting the first two numbers from the last one. Regarding the PAVN module, the total elapsed time is 0.34 second, the user process takes 0.28 second, and the system time consumes about 0.02 second on file operation. We therefore saved about 1 second on the server side by using PAVN instead of the CC/PP module.

In our novel implementation, several issues are not considered. First, the CC/PP file is fetched from the proxy server which certainly saved a lot of time. Second, in the real implementation of CC/PP, there are always some delta data sent to the Web server from the client as complementary information, for example, the upgraded operating system version or the size of the memory sticks which was just inserted into the expanded slots. The extra information will also cost some extra time for the CC/PP module. Third, we used an average size of CC/PP files, which is 4.24k bytes because we could not find out a standard CC/PP file for the Motorola i95cl phone [33].

The processing time of the Versioning Selection Algorithm (VSA) on both server and phone is insignificant compared with the object delivering time spent on the wireless and wired networks. With a lack of benchmark tools on cellular phones, we could not find out the accurate time spent on the phone client as we did on the UNIX server. But we got several timestamps while our application was running. It takes only 25ms to parse the fidelities of one object as shows. But we use only this datum as a reference since it is measured inside the program and thus not accurate.

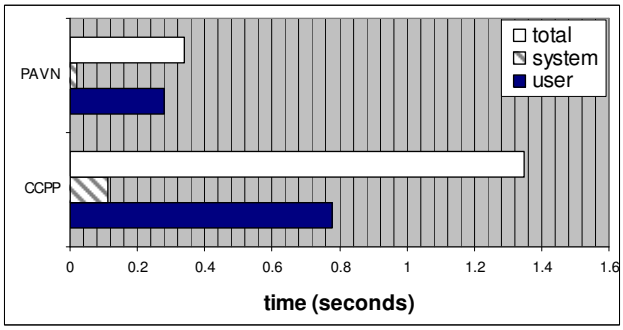


Figure 18. Time measured on the server side

It is interesting to observe that there is a significant delay on setting up a connection on a wireless network. The experiments show three cases to download 512k Web objects: 256+256, 512, and 512+0. In PAVN, to download 512 bytes with one round-trip consumes 5398ms. But the others take 6959 and 6677ms separately. It is similar to the CCPP method. Therefore, it would be more efficient to download everything necessary at once on the current

wireless network, which is similar to the WAP solution. But pushing Web contents is a preemptive solution which is opposite to our pulling method in versioning negotiation.

Finally, we can see the trend of computing power of small devices and their connected wireless links will be developed extremely fast in the near future. The saved time by deploying PFML will be more significant. Moreover, we do not need different solutions to different devices. The ubiquitous Web will finally appear on all devices.

5. Conclusion

We have presented two approaches to adaptive web delivery to end users with diverse mobile devices. The first approach is *partiality adaptation*, which uses priority tags to enable both communication ends to indicate their intentions and allow different device users to share the same web pages. The second approach is *versioning negotiation,* which uses fidelity tags to let client agents make their decisions to choose the best available version without incurring an extra round of message exchange.

The primary focus of our future work is to find out the threshold of network speed that can significantly offset the benefits of our approaches on a wireless network based on our experiments.

6. Reference

[1] RFC 2296 Remote Variant Selection Algorithm.

[2] F. Douglis, A. Haro, and M. Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. *In USENIX Symposium on Internet Technologies and Systems, Monterey, California, USA , Dec. 1997. USENIX Association*.

[3] Edge Side Includes, http://www.esi.org.

[4] C. Chi and Y. Wu . An XML-Based Data Integrity Service Model for Web Intermediaries. *In Proceedings of the 7th International Workshop on Web Content Caching and Distribution, Boulder, Colorado , August 2002.*

[5] F. Reynolds, J. Hjelm, S. Dawkins, and S. Singhal, "Composite Capability/Preferences Profiles: A user side framework for content negotiation," *27 July 1999.*

[6] Gimson, R., et. al., "Device Independence Principles", *W3C Working Draft, September 2001.*

[7] C. Ng and P. Tan . QoS and Context Delivery in Rule-Based Edge Services. *In Proceedings of the 7th International Workshop on Web Content Caching and Distribution , August 2002.*

[8] W. Shi and V. Karamcheti. CONCA: An Architecture for Consistent Nomadic content Access. *In Proceedings of the WCCC Sorrento, Italy , June 2001.*

[9] Wireless Markup Language(WML)

[10] V. K. and A. Joshi. "An End-End Approach to Wireless Web Access," *In Proceedings of the International Workshop on Wireless Networks and Mobile Computing, April 2001.*

[11] Transparent Content Negotiation in HTTP. [RFC2295]

[12] A Syntax for Describing Media Feature Sets. http://www.faqs.org/rfcs/rfc2533.html

[13] M. H. Butler Current Technologies For Device Independence. *Hewlett Packard Laboratories Bristol; External Technical Report HPL-2001-83; 30 March 2001*

[14] Smith, J.R., Mohan, R. & Li, C.. Transcoding internet content for heterogeneous client devices. In *Proceedings of IEEE Conference on Circuits and Systems , 1998*

[15] Bjorn Knutsson, Honghui Lu, and Jeffrey Mogul. Architecture and Pragmatics of Server-Directed Transcoding. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, Boulder, Colorado , August 2002.

[16] Chi-Hung Chi and Yang Cao . Pervasive Web Content Delivery with Efficient Data Reuse. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, Boulder, Colorado , August 2002.

[17] Intermediary Rule Markup Language (IRML) *http://www.ietf-opes.org/documents/draft- beck-opes-irml-02.txt*

[18] RFC 2005

[19] Dirk Husemann, Pervasive Computing *Computer Networks Volume 35, Issue 4, March 2001*

[20] Mark. Weiser, The computer for the twenty-first century, *Scientific American, September 1991*

[21] Davison, B.D. A Web caching primer *Internet Computing volume 5 number 4 July/August 2001*

[22] T. Lemlouma, N. Layaida. Content Adaptation and Generation Principles for Heterogeneous Clients. *OPERA Project, INRIA Rhone Alpes.*

[23] http://one.cs.washington.edu/csharp/network.html.

[24] A. Fox and E. A. Brewer: Reducing WWW latency and bandwidth requirements by real-time distillation. *Proc. of the 5th International 3WC*, Paris, France (1996).

[25] SF Chang, "Optimal Video Adaptation and Skimming Using a Utility-Based Framework," *in Proc. IWDC-2002 , Capri Island, Italy, Sep. 2002.*

[26] Z. Lei, N. D. Georganas, "Context-based media adaptation for pervasive computing", *Proceedings of Canadian Conference on Electrical and Computer Engineering Toronto, May 2001.*

[27] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. "An Active Transcoding Proxy to Support Mobile Web Access," In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998.

[28] The Convergence of Heterogeneous Internet-Connected Clients Within IMASH. *IEEE Wireless Communications 06'02.*

[29] Bjorn Knutsson, Honghui Lu, and Jeffrey Mogul, Architecture and Pragmatics of Server-Directed Transcoding, *Proceedings of the 7th International Web Content Caching and Distribution Workshop, pp. 229-242, August 2002.*

[30] A. Joshi. On Proxy Agents, Mobility and Web Access, Baltzer Mobile Networks and Applications 5, 2000

[31] I95cl Multi-Communication Device J2ME Developers' Guide, Motorola Inc. 2002.

[32] Phone description, available: http://www.ilmunc.org/motorola_i95cl.html

[33] Device profiles, available: http://w3development.de/rdf/uaprof_repository/ , W3C 2001.