Client-Side Web Acceleration for Low-Bandwidth Hosts*

Tae-Young Chang†, Zhenyun Zhuang‡, Aravind Velayutham§, and Raghupathy Sivakumar†

†School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
‡College of Computing, Georgia Institute of Technology, Atlanta, GA 30332
§Asankya Inc., Atlanta, GA 30308
Email: key4078@ece.gatech.edu, zhenyun@cc.gatech.edu, vel@asankya.com, siva@ece.gatech.edu

***Abstract*—Current popular web-browsers simply fetch the entire web-page from the server in a greedy fashion. This simple web fetching mechanism employed by browsers is inappropriate for use in low-bandwidth networks, since they cause large response times for users unneccesarily. In this paper, we first analyze the reasons that cause large response times by considering several factors including the properties of typical web-pages and browsers, the interaction of the HTTP and TCP protocols, and the impact of server-side optimization techniques. We then propose three easy-to-deploy browser-side optimization mechanisms to reduce the user response time. Through simulations, we compare the performance of our solution with that of current browsers and show that the proposed scheme brings significant performance benefits in terms of user-perceived response times.**

I. Introduction

In the past couple of decades, tremendous amount of research has been done on improving web access performance over Internet. Optimization techniques such as caching web-proxies ([1], [2], [3]), persistent HTTP connections ([4], [5]), and content distribution networks ([6], [7]) have found wide-spread adoption. New paradigms such as WAP[8] and BREW[9] have also been developed to address web performance limitations on mobile hosts.

In this work, we study the performance of a web-browser under low-bandwidth network conditions. Specifically, we analyze the characteristics of a web-browser with the objective of identifying reasons as to why they might suffer in low-bandwidth conditions. We find that the current fetching model employed by commercial web-browsers is not optimal in bandwidth challenged environments.

We show that the absence of content prioritization and intelligent object fetching mechanisms in current web-browsers leads to increased response times. Browsers, today, do not prioritize the useful data that is viewed by the user over other data in the web-page. With a greedy fetch of the entire content of a web-page, precious bandwidth is wasted and in turn increases user perceived response time. Further, without intelligent object fetching mechanism, the download process of current web-browsers does not utilize the network bandwidth efficiently.

To make this problem even worse, many web-pages have become larger both in pixel- and byte-size with a large number of external objects. For example, `cnn.com` has a main page, which is larger than 3 times the pixel-size of *client area* that is defined as content area within a main browser window, and greater than 300-KB data, with hundreds of external objects. Thus, even with a network with bottleneck bandwidth being 100 Kbps, the time taken to fetch the entire page can be larger than 20 seconds.

In this paper, we propose a client-side only solution to address the problem of large response time with current browsers. The solution is based on careful consideration of several factors including the content displayed on the screen viewed by the user, server-side content distribution networks, and the relationship between the HTTP and TCP protocols. The three mechanisms we propose include prioritized fetching, object reordering, and connection management. One major advantage of our approach is that it is pure client-side enhancement and does not need any server changes. The proposed solution helps to reduce response time, and is easy to deploy since it only requires client-side installation to current web-browsers.

To summarize, our contributions in this paper are:

- Identification of the inefficiencies of current web browsers by carefully analyzing the interactions of several factors related to web fetching.
- Proposal of three mechanisms to reduce the user response time in an easy-to-deploy fashion.

The rest of the paper is organized as follows. Section II discusses the problems with current web access models and presents the impact of problems. Section III presents the details of our solution. Section IV evaluates the performance of our proposed scheme with that of conventional web access models using simulations. Section V discusses related works in the area, and Section VI concludes the paper.

II. Model and Motivation

In this section, we describe drawbacks in the conventional web access model in low-bandwidth environments and use them as motivation for designing a new web access scheme.

A. Web Access Model and Simulation Setup

1) Web Access Model: The conventional network model for web access is as shown in Figure 1. In this model, in order to access a web page, a user feeds in an URL address. Then, the browser requests a DNS server to translate the URL address into an IP address. After obtaining an IP address of the corresponding web-server from the DNS server, it requests the HTML document from the web-server directly. When load

*This work is supported in part by the National Science Foundation under grant CNS-0519733, and the Georgia Tech Broadband Institute (GTBI).

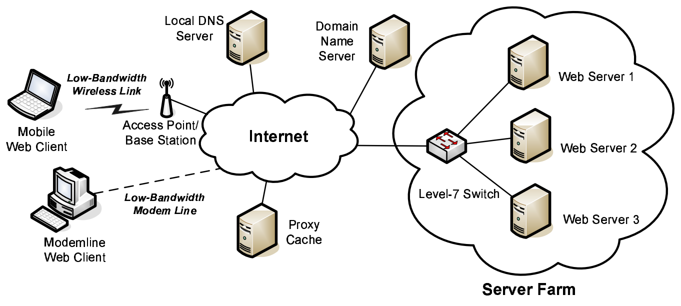


Fig. 1. Network topology

TABLE I
WEB CHARACTERISTICS OF TOP 50 WEB SITES

		HTML	IMG	Others	Total
Byte-size per object [KB]	Mean	31.72	2.46	12.91	225.96
	STD	35.51	5.90	9.67	186.04
Number in first screen	Mean	1	17.31	4.41	22.72
	STD		15.36	6.22	16.37
Number in all screens	Mean	1	46.80	3.99	51.79
	STD		28.16	6.22	30.34
Number of web-servers	Mean	1	5.16	1.74	5.50
	STD		2.90	1.20	3.38
Width [pixels]	Mean				998
	STD				46.49
Height [pixels]	Mean				1937
	STD				1119

balancing is performed among multiple servers (e.g. a server farm), a layer-7 switch rewrites domain names used in the HTML document in order to distribute requests of embedded objects to multiple servers. Finally, the web-browser performs DNS resolution for other unknown web-servers and downloads objects from them.

Typical web-browsers open multiple connections to a single web-server in order to increase fetching speed. For example, Internet Explorer and Netscape Navigator open up to 2 and 6 connections to a single server respectively[10]. A parsing engine in the browser inserts object requests to message queues of the multiple connections in a round-robin fashion because the browser is unaware of object and network characteristics.

In this paper, we consider the *Top 50 Web Sites*[11] as representatives of typical web pages, and measure their web characteristics. Table I shows the results of the measurement. In the table, *screen* refers to an effective area for displaying a web page in the browser. When we set the default screen resolution as 1024-by-768, the pixel-size of the client area in Internet Explorer is 1006-by-511, and we set the size of a screen as this value. The initial screen is the first part that is shown in the client area when a web page is accessed. Therefore, the average number of screens in these pages is $\frac{1937}{511} = 3.7$.

2) Simulation Setup: In order to evaluate the performance of the conventional web access models in low-bandwidth environment, we use *ns2 simulator*[12] with the *Reno-FullTCP* package which support bi-directional transmissions. In the simulations, the same network topology as shown in Figure 1 is used with the assumption that a local DNS server has all the

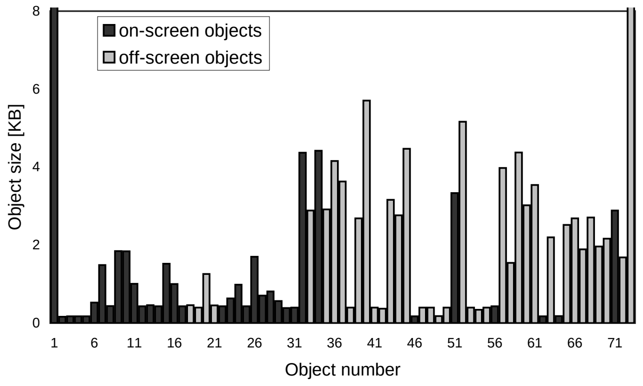


Fig. 2. Object fetching sequence at `amazon.com`

required domain information. The bottleneck link is located between the web client and the backbone network, and is configured to have 100-kbps bandwidth and 100-ms link delay. The bandwidth and delay from both DNS and web-servers to backbone networks is 1 Mbps and 5 ms respectively.

For modeling web traffic, we use the same web characteristics as shown in Table I. The average processing time of each object in the browser is assumed as 200 ms, and parsing delay is ignored. We assume that all web servers support HTTP/1.1 with the persistent connection feature, but pipelining is not considered since it is not faithfully supported by most commercial web-servers[13]. The size of a HTTP request message is 500 bytes, and the size of HTTP reponse header is ignored. We also assume that the cache function of the browser is disabled.

We consider *initial screen response time*, which is defined as the difference between the time when a web-browser sends a request for a HTML document and the time when all objects for displaying the initial screen are downloaded completely, as a primary metric.

B. Screen Contention Problem

When a user is viewing a screen on a display device, objects for displaying other screens are unnecessary in the sense that they are not visible to the user at this time. However, in conventional web browsers, the process of fetching *necessary* on-screen objects (*i.e.* objects on current screen) may be slowed down due to the competition from the process of fetching *unnecessary* off-screen objects. We refer to the fact that objects from different screens are competing for bandwidth as *screen contention*.

The main reason of screen contention is disparity of cumulative transfer-size among multiple connections. As mentioned earlier, a parsing engine inserts object requests to message queues in a round-robin fashion that considers only fairness in terms of the number of objects per connections. As a result, some connections having only small-size objects may finish transmissions of on-screen objects early and begin to fetch off-screen objects. Under this scenario, different connections may fetch objects on different screens simultaneously.

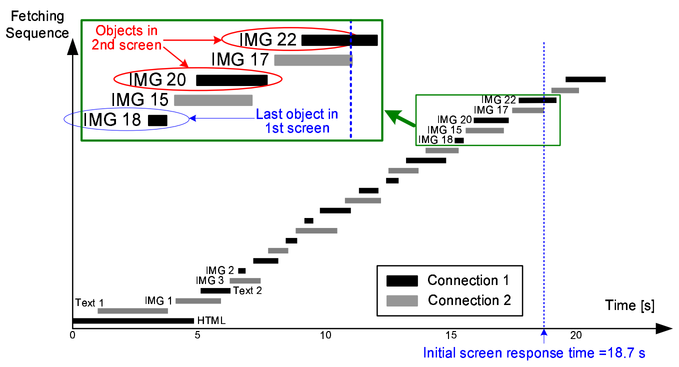


Fig. 3. Fetching with screen contention in conventional web-browsers

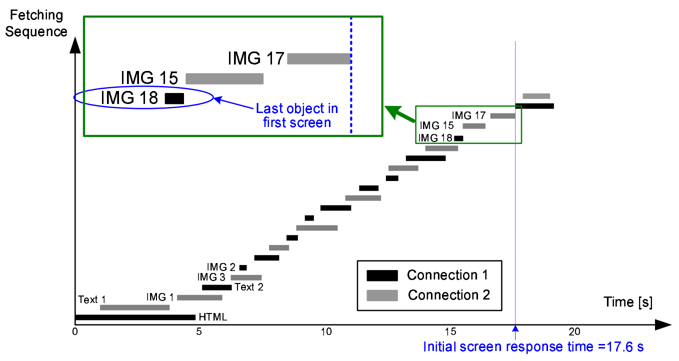
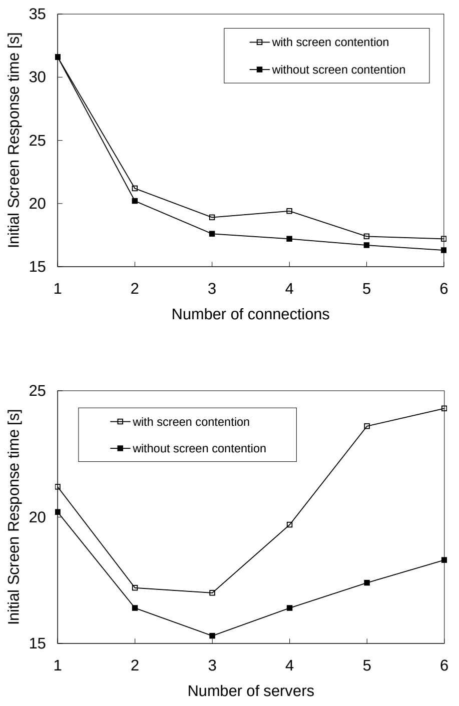


Fig. 4. Fetching without screen contention in ideal web-browsers



(b) Multiple server case

Fig. 5. Impact of screen contention

Another possible reason is directionality in a table structure. When a multi-cell table is defined in a web page, one of internal cells may have larger vertical pixel-size than the client area in the browser window. In such a case, a web browser fetches *off-screen* objects located at the end of this cell first, and then begins to fetch *on-screen* objects at the beginning of the next cells. Figure 2 shows an example of the object fetching sequence in Internet Explorer[14] at the `amazon.com` page, which consists of 1 HTML document, 5 javascript, 2 flash, and 65 image objects. In the figure, most off-screen objects are fetched or begin to be fetched before all the 35 on-screen objects are downloaded because of fetching directionality in the table.

We show the effect of the screen contention problem by presenting the simulated object fetching progress in Figure 3. We assume that all the objects are from a single server, and the effect of directionality in a table structure is ignored. In the simulations, the initial screen has 18 objects, *i.e.* objects numbered after 18 are unnecessary data. As observed from the figure, objects from both the current screen and other screens are fetched simultaneously due to the screen contention problem. Since fetching the unnecessary objects consumes some portion of bandwidth, the resulting response time for initial screen is increased unnecessarily. As shown in the figure, image (IMG) objects numbered 20 and 22 are fetched in parallel with other objects on the initial screen. As a result, the response time of initial screen, which is measured after IMG 17 is downloaded, is 18.7 s.

An intuitive solution to screen contention is to prevent unnecessary object fetching. Figure 4 shows an ideal case that contention is eliminated in an ideal browser. As seen from the figure, when the faster connection (Connection 1) completes the downloading of all the objects on the initial screen, it stops fetching and waits for the other connection (Connection 2) to finish fetching the objects on the initial screen. By doing so, the remaining connection can obtain more bandwidth and in turn reduces the response time for the initial screen by 1.1 s.

Moreover, under scenarios where content of a web page is distributed over multiple servers, this performance degradation becomes even worse because the degree of contentions among connections to different server becomes higher. Figure 5(a) shows how screen contention affects the initial screen response time under both single and multiple servers scenarios with varying number of connections and servers. In the multiple servers case, we assume that up to 2 parallel connections are allowed to each server.

For a web-browser to fetch objects from a single server, it can be seen that as the number of connections increases from 1 to 3, both the conventional and ideal browsers show significant performance improvement. However, when the number of connections is larger than 4, the performance is less affected by it. In such scenarios, the ideal browser shows less performance improvement.

When web objects on a single page are from multiple servers by some load balancing techniques, the performance is directly affected by the number of servers. In Figure 5(b), both the schemes show the best performance as the number of servers becomes closer to 3. However, the performance in

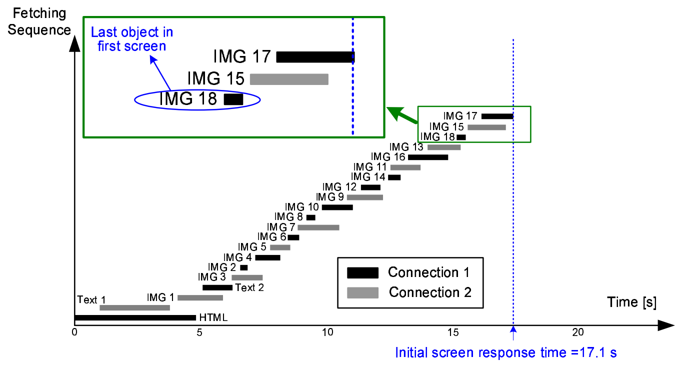


Fig. 6. Fetching with synchronized ending time in ideal web-browsers

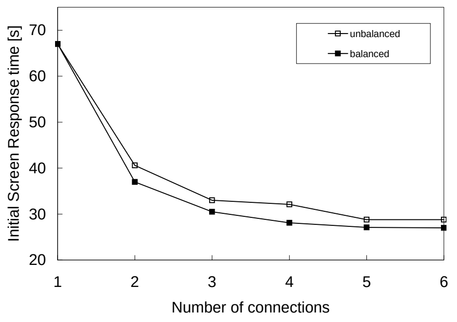
the ideal scheme is less influenced by the number of servers due to its effective prevention of screen contention.

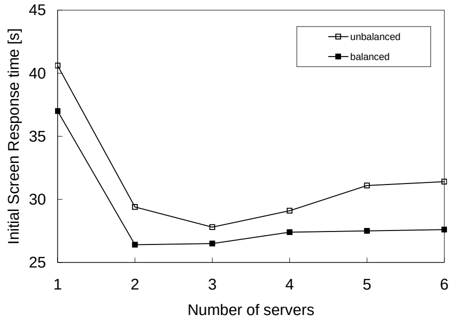
C. Bandwidth Under-utilization Problem

In HTTP/1.1, a persistent connection consists of a series of request-response transactions. Given this model, [13] shows that the idle time of the network decreases with the increase in the number of simultaneous TCP connections. The authors of [13] also show that there is an optimal number of simultaneous TCP connections (around 6) at which the performance is optimal because of reduced idle time. However, since current web-browsers do not schedule object transfers in a bandwidth-efficient way across multiple TCP connections, current web transfers do not always maintain the optimal number of simultaneous TCP connections. In many cases, only a small number (e.g. 1 or 2) of connections are active at any instant. This results in under-utilization of the access link which we refer to as the *bandwidth under-utilization* problem.

The above-described bandwidth under-utilization problem results in varying levels of performance degradation depending on specific server scenarios. In the case of a single server, bandwidth efficiency is determined by synchronized ending times of transmission among parallel connections. In Figure 4, the last object, IMG 17, is fetched with no other objects, and thus only a single connection uses network bandwidth toward the end. In cases of multiple servers, the user performance is also affected by synchronized ending among connections to different servers.

The solution to the bandwidth under-utilization problem is to schedule the GET requests across the multiple TCP connections such that all the TCP connections are always active during the fetching process. Figure 6 shows the impact of performing ideal scheduling such that there is a pending request in each of the TCP connection. In the figure, when the faster connection finishes fetching all the objects in its request queue and has no more objects to fetch, it takes over the unfulfilled object requests from the queue of the other connection and perform fetching. As a result, both the connections can use bandwidth more efficiently and improve the initial screen response time by 1.6 s when compared to conventional web-browsers.





(b) Multiple server case

Fig. 7. Impact of bandwidth under-utilization

Significant amount of web content is being served from distributed web-servers. Ease of content update and server load-balancing are some of the benefits of employing multiple web-servers to serve web content. In the scenario of multiple web-servers, different objects belonging to the same web-page are delivered by opening TCP connections to the different servers. Commercial web-browsers do not take into account the size of the objects in scheduling the different object requests. This invariably leads to scenarios where several TCP connections to different web-servers are idle while the other connections are active.

The intuitive solution is to schedule the different object requests across the multiple servers such that all the connections are active. Figure 7 shows how bandwidth under-utilization affects the response time performance under various server scenarios. Note that the response time in the figure means the transfer time for all the objects of a web page, and thus screen contention does not exist in this scenario. In Figure 7(a), the single server case shows a similar pattern as in Figure 5(a) and improves the performance consistently in the entire range. In Figure 7(b), unlike Figure 5(b), as the number of servers increases beyond 3, both the schemes show stable or slightly worse performance. In this figure, the ideal scheme shows up to a 20% performance improvement.

D. Summary

We have identified two issues with conventional web-browsers in bandwidth-limited networks. We observe that contention among objects belonging to different screens within the same web-page can increase user-perceived response time of the initial screen. We also identify that network bandwidth can be under-utilized because of two issues with conventional browsers. observe that in most cases screen contention and bandwidth under-utilization problems affect user performance negatively in a conventional web model, and show how the ideal browser can overcome these problems and achieve significant performance improvement. Based on these observations, in the next section, we propose a new web access scheme.

III. Solution

A. Overview

Our proposed solution includes three mechanisms, namely prioritized fetching, objects reordering, and connection management. The brief summary of the mechanisms is as follows.

- *Prioritized fetching (PF)* addresses the screen contention problem in a multiple-screen page and provides an optimization solution for fetching objects with varying priority levels. Basically, PF is a *What-You-See-Is-What-You-Fetch (WYSIWYF)* mechanism, and while giving higher priority to the on-screen objects, while it gives lower priority to the off-screen ones in order to reduce the user perceived response time.
- *Object reordering (OR)* addresses the bandwidth under-utilization problem when fetching objects from the same server. When load on connections to a single server is unbalanced, OR reschedules object requests across connections.
- *Connection management (CM)* addresses the bandwidth under-utilization problem when multiple servers are involved in a web-page. In order to balance load among connections to different servers (*i.e.* domains), it performs dynamic re-assignments for entire connections.

The three mechanisms complement each other as well as performing optimization with different levels of granularity for web object fetching on the web-browser side. One of the advantages of our solution is easy deployment, since it requires only client-side modification. In fact, the solution can be implemented as nothing more than an add-on to the current web-browsers. Figure 9 shows the flow charts of these mechanisms, and Figure 8 shows where they are located in the entire data flow.

B. Prioritized Fetching (PF)

Conventional web-browsers begin to fetch and parse objects as soon as finding definitions of objects while downloading a HTML document. This on-the-fly fetching mechanism may bring performance improvement in high-bandwidth networks because the overhead of screen contention is relatively small in this environment. However, in a bandwidth-limited client, this overhead affects user performance significantly as mentioned earlier. Thus, PF differentiates objects from different screens based on the current screen view and allows for downloading only the objects that are required to render the current screen display. As a result, it reduces the response time experienced by web users.

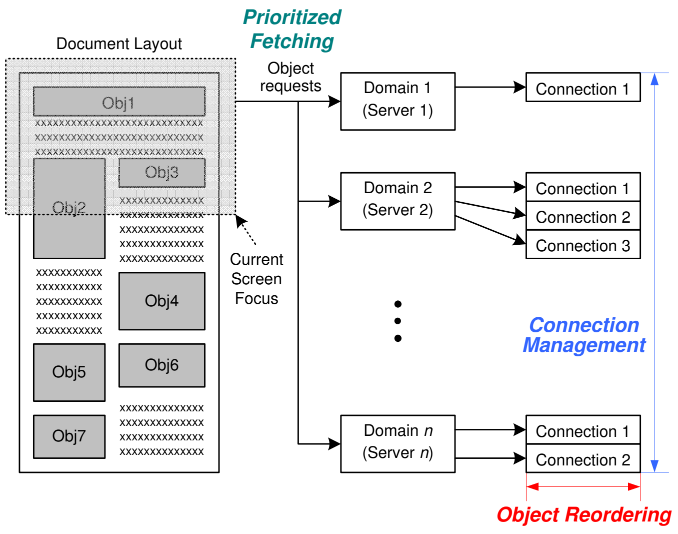


Fig. 8. Overview of three mechanisms

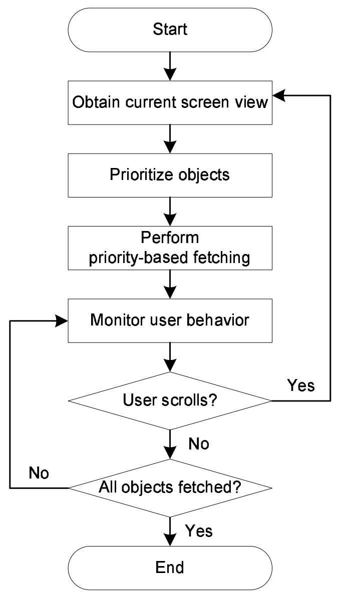
The basic operation of PF is as following: (1)When a web access is performed, PF first obtains the initial screen view information in the entire document layout and prioritizes embedded objects according to their locations in the document layout. (2)Then, it performs fetching objects according to their priority levels. (3)When a user scrolls to move to a different view, PF performs the above-mentioned process again for the new screen.

PF consists of three components, initial object prioritization, selective object fetching, and re-prioritization for screen update. The detailed operations are illustrated in Figure 9(a).

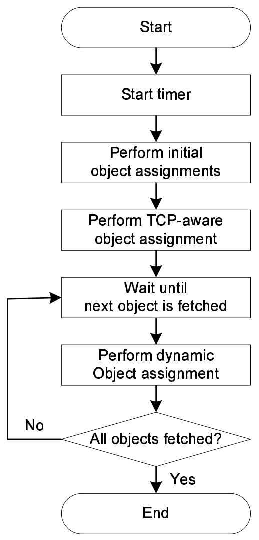
1) Initial Object Prioritization: Generally, a web-page consists of various types of embedded objects. PF considers text-based files including HTML, javascript, cascading style sheets, and other layout-related files, as the highest-priority objects since these objects play an important role to construct the overall HTML display layout. On the other hand, for other types objects such as image and multimedia objects, PF assigns different priority levels according to their locations. For simplicity, we consider only IMG objects as representatives of objects that do not affect the document layout.

As mentioned earlier, PF performs location-based prioritization for IMG objects. The detection of pixel-location of an IMG object is possible because most HTML document files defines the pixel-size of image objects and a web-browser can construct the full page layout without downloading these objects. In cases that the HTML document does not specify the pixel-size of an image object that is not fetched yet, PF uses an pre-obtained averaged value based on browsing history.

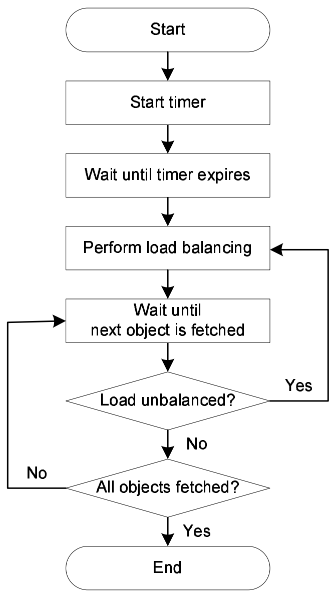
In order to get the location information of objects, PF scans the document object model (DOM) tree[15]. When it finds an IMG object definition, it searches all the successors in the tree and calculates location offsets from successors to predecessors in a recursive fashion until it reaches the top of the tree. The absolute location in the layout is defined as the sum of all the relative offset. Based on this location information, PF gives



(a) Prioritized fetching (PF)


Start
Start timer
Perform initial object assignments
Perform TCP-aware object assignment
Wait until next object is fetched
Perform dynamic Object assignment
All objects fetched?
No
Yes
End

(b) Object reordering (OR)



(c) Connection management (CM)

Fig. 9. Flow charts of three mechanisms

highest priority level to objects that are located within the current view in the client area and low priority levels to others.

2) Selective Object Fetching: For the schemes used to fetch objects of different priority levels, many existing schemes such as [17] allocates a small portion of bandwidth for low-priority transmissions. However, these schemes cannot be efficiently exploited in PF due to the following reasons: (1) HTTP 1.1 defines persistent connections, which allows transfer of multiple objects using a single TCP connection. In PF, a priority-based connection doesn't have flexibility to send both high- and low-priority objects. (2) Each TCP connection performs multiple short transmissions. As mentioned earlier, generally IMG objects that account for the half of total byte-size of a web-page have a small average byte-size (a few packets). In these short bursty on-off transmissions, priority-based connection schemes can't assign a desired portions of bandwidth accurately.

Thus, PF uses a delayed-transmission scheme. When information of new objects are extracted from a HTML document and they are prioritized as high level, PF inserts the corresponding request messages into the already-in-use queues. In this scheme, low-priority objects are fetched only after all the higher-priority objects have been downloaded, *i.e.* after the higher-priority queues become empty.

3) Re-prioritization: When browsing web-pages, a user may scroll to another view other than the current one before all the on-screen objects in the current screen are fully downloaded. For example, a user may perform *fast scroll* by searching and clicking an internal link to another part in the same page. In these scenarios, the current focus is changed before the downloading of previous screen, and the initial prioritization may not perform efficiently. Thus, a proper mechanism is required to deal with these scenarios.

When the screen focus is moved to a new area, PF removes all the IMG objects that reside in request queues and re-prioritizes them for the newly focused area. For the objects that are currently being downloaded, PF waits for their completeness. The reason for allowing this off-screen fetching is that most web-browsers, as applications above transport layer following HTTP standards, do not have mechanisms to manage disconnections and re-connections. PF thus keeps the currently incoming transfer and only updates the priority levels of the queues involved.

The provided fetching schemes can be different transport protocols, different parameters in the same protocol, or different starting time. As mentioned earlier, in this work, we consider only adjusting starting time to fetch different screens using the currently existing transport protocols.

C. Objects Reordering (OR)

For parallel connections to a single server, OR uses balanced ordering of objects to gain benefits in terms of reduced response time. The operations of OR consist of three steps. (1) At the first step, an initial assignment of objects is executed. (2) Then, an optimized ordering of objects is performed by TCP-aware object reordering. (3) After that, it performs dynamic objects rescheduling until all objects are completely fetched. The detailed operations are illustrated in Figure 9(b).

1) Initial Objects Assignment: As we identify in Section II, conventional browsers perform a round-robin assignment to distribute object requests to multiple connections. This size-unaware assignment may cause unsynchronized ending time among different connections, and as a result increases response time. Therefore, OR performs load balancing among connections using byte-based metric rather than simple round-robin in order to synchronize their ending time.

Since larger byte-size translates to longer downloading time in web fetching, OR synchronizes ending time of different connections by distributing same amount of objects to every connection. A more accurate way to synchronize the ending time could be one that also considers the number of objects, the precessing time for each object, and others. That is, the expected ending time is given by $\frac{Size_{Data}}{BW_{Available}} + n * \frac{rtt}{2} + T_{Proc}$, where n is the number of objects and T_{Proc} is the processing time.[1] However, OR simplifies the metric by considering only data size, based on the observation that the first term, $\frac{Size_{Data}}{BW_{Available}}$ dominates over other terms in low bandwidth networks.

Performing OR requires the byte-size information of objects. Since this information is normally not included in HTML documents, OR estimates it by considering both the object's pixel-size included in HTML documents and the object formats such as gif and jpeg. Based on this data size information, OR sends object requests through multiple connections in a balanced way. A time with a φ expiration value is used to strike the balance between amount of objects and increased response time.

2) Dynamic Objects Rescheduling: Irrespective of initially balanced assignment of objects among connections, due to dynamic behavior of connections, the total fetching time of different connections may still vary significantly. If due to some reasons one connection is delayed, and the other connection is idle, it is possible to reschedule the objects from the busy connection to the idle one, and thus reduce response time even more. Dynamic objects rescheduling runs in an on-demand fashion during the fetching process in order to deal with the dynamic nature of the connections.

3) TCP-aware Objects Reordering: When initial objects are assigned to connections, TCP-aware reordering of fetching sequence can minimize the adverse effect of slow-start in TCP connection setup and increase fetching speed. For example, let us assume there are 3 objects with data size of 7, 3, and 2 KB respectively are waiting to be fetched along one single connection. If the connection is newly created, the total fetching time for an order of 7→3→2 KB may be 5 rtts since it takes up to 3 rtts to fetch the first object that has 7 KB data size and one more rtt for each of two other objects.[2] However, if an opposite order of fetching is allocated, it may just take 3 $rtts$ to finish the fetching. Thus, appropriate ordering the fetching may save response time in the order of rtt of the path.

TCP slow start can kick in at any time during the downloading process. However, since HTTP and upper layer is unaware of each other's status information, there is no way to take advantage of them without some other cross-layer mechanisms. Thus, the TCP-aware objects reordering scheme only makes use of the slow start phase in the beginning of a TCP connection.

[1]Since fetching each object has to follow the HTTP request-reply handshaking pattern, which wastes 0.5 rtt.

[2]More than 95% of the servers do not perform TCP-JumpStart[16], and we assume that the initial value of congestion window in web-servers is 1.

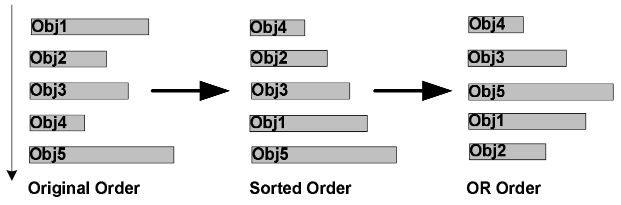


Fig. 10. Reordering using OR

For this rescheduling to take place, appropriate ordering of objects is required. Intuitively, with small objects being put at the end of connection, it is more likely to reschedule objects among connections, and thus reduce response time. Also, ordering objects from big to small also makes rescheduling easily to perform, since small objects can be rescheduled in a *finer* granularity as the fetching process going on.

Thus, both considering the two requirements, OR orders the fetching sequence in a *rats-elephants-rats* fashion. The detailed operation of TCP-aware Objects Reordering is illustrated in Figure 10. First, all the objects assigned to one connection are sorted according to their data size. After that, from smallest one, all objects are inserted from two ends of the queue in round-robin way, and the resulting ordering is a small-to-big-to-small order.

D. Connection Management (CM)

CM addresses the bandwidth under-utilization problem when fetching objects from multiple servers by controlling the numbers of connections a browser can open to different servers. By adjusting the number of connections for each server, CM effectively synchronizes the ending time of downloads in the connections. As a result of the improved bandwidth utilization, the response time is reduced. CM consists of two components, estimation of per-connection load and dynamic connection assignment. The detailed operations are illustrated in Figure 9(c).

1) Per-Connection Load Estimation: In order to estimate the ending time of downloading, CM uses the byte-size information that OR converted earlier. The intuition of CM is to assign more connections to servers with larger data size, while assign less connections to servers with smaller data size. To maintain friendliness to current browsers and compatibility to published standards, the total number of connections in our mechanism is maintained the same as in today's popular browsers. By doing so, CM behaves *friendly* to them. To achieve this purpose, whenever it assigns one more connection to some server, one less connection should be deducted from some other server. Furthermore, CM limits the maximum number of connections assigned to a server to four due to several reasons including the observation made by [13] stating that allocating too many (say, more than 6) connections to the same server does not necessarily lead to better performance.

2) Dynamic Connection Assignment: When fetching a HTML document from a server, a browser fetches and parses the contents. Whenever it detects new object information, it estimates the byte-size of this object, and starts a timer with δ expiration value. The setting of this timer requires careful consideration. On one hand, CM needs to collect some amount of object samples in order to achieve improvements. Thereby the δ should not be too small. On the other hand, CM should not delay object fetching significantly to avoid increasing response time adversely, and thus the expiration value should not be very large such as dozens of ms.

After the expiration of this timer, CM performs the initial assignment based on object information collected so far. During the process of fetching objects, it keeps recording the object information on how much data already received. This information will be used again to adjust the number of connections

3) Mathematical Model: This CM mechanism can be formulated into the following mathematical model. Given a set of servers, $S = \{s_i | 1 \leq i \leq N\}$, where N is the total number of servers, let d_i denote the total data size of objects from server s_i. Given a connection set, $C = \{c_i | 1 \leq i \leq N\}$, where c_i is the number of connections opened for server s_i, CM finds a minimized maximum value of d_i/c_i.

C is also subject to three other constraints. First, the total number of connections should not exceed $2N$ to maintain friendliness to current browsers. Second, the c_i should not exceed the number of objects in s_i. Third, c_i should have a range from 1 to 4.

Let us use n_i to denote the number of objects in server s_i. The output C should achieve the purpose described in Eq. 1, and satisfy the constraints denoted in Eq. 2.

$$L_{max} \textit{ is minimized}, \quad \textit{where } 1 \leq i \leq N \tag{1}$$

$$\sum_{1 \leq i \leq I} C_i \leq 2I \quad \textit{and} \quad 1 \leq C_i \leq min\{4, N\} \tag{2}$$

The detailed algorithm is as follows. In the beginning, every server is assigned 2 connections, and CM computes the largest and smallest values of $L_i = d_i/c_i$. We use L_{max} and L_{min} to denote the largest and smallest values of L_i. Assume server s_j has the largest value (*i.e.* $L_{max} = d_j/c_j$), and server k has the smallest value, L_{min}. Now, we increase c_j by 1 (*i.e.* $c_j = 3$), and decrease c_k by 1, then compute the new maximum value, $L_{max'}$. If $L_{max'} < L_{max}$, that means the new assignment has a smaller maximum value, then the algorithm will continue to run. Otherwise, if $L_{max'} > L_{max}$, the algorithm stops and resume to previous assignment, since the new assignment results in a larger maximum value.

The algorithm runs whenever an object is downloaded. For the new runs, the metrics considered in Equation 1, *i.e.* s_i, are set to be the remaining data size for server s_i. Thus, the algorithm will be performed whenever an object-related event happens. For this reason, an adverse effect - fluctuation on the number of connections assigned to servers, may possibly happen. To reduce this fluctuation, we introduce a threshold value, τ. In CM, only when the $L_{max'}$ values between two consecutive iterations is larger than τ, the algorithm will adjust connections set. We suggest a 10% of previous $L_{max'}$ as the τ value.

IV. Performance Evaluation

In this section, we evaluate the performance of the proposed mechanisms, and compare it with that of conventional web-browsers.

A. Simulation Setup

In order to evaluate the performances, we use *ns2 simulator*[12]. Unless otherwise noted, the network configurations as well as the web characteristics used in simulations are the same as described in Section II. We use the same network topology as shown in Figure 1 with the assumption that the local DNS server has all the required domain information.

Response time for the initial screen is used as the primary metric for comparing performances. In this section, we compare five schemes including conventional (CONV), PF only (PF), PF with OR (PF+OR), PF with CM (PF+CM), and all integrated (ALL) schemes. To better explore the impacts of some factors on the performances, we vary some factors in evaluation. These factors include object characteristics such as object sizes and total number, numbers of servers involved, number of connections opened to a single server, network characteristics such as link bandwidth and rtt, and user's fast scrolling to different screens.

B. Impact of Object Characteristics

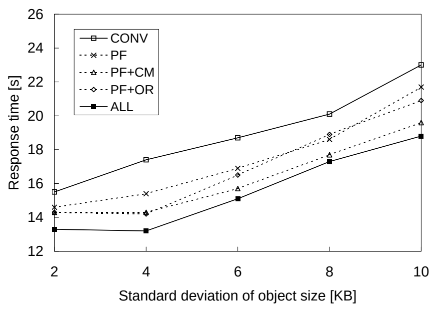
Figure 11 shows the initial response time of conventional web-browsers, and our proposed solution when the standard deviation of individual object size and the number of objects in the first screen vary. As shown in the figure, when used in combination, the three proposed mechanisms can reduce up to 30% of response time compared to current browsers.

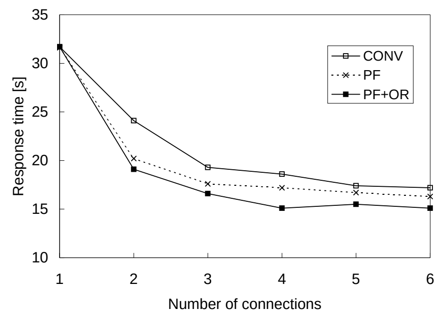
Figure 11(a) show that as the variance of object size increase, the performance of both the conventional model and our scheme shows worse performance. For conventional web-browsers, the reason is obvious, since larger variance can be translated to reduced bandwidth utilization as described in Section II. Our mechanisms can alleviate this problem, and thus reduce the initial response time. However, since the problem still exists, and becomes more severe when variance of object size increases, the performance degradation is still expected.

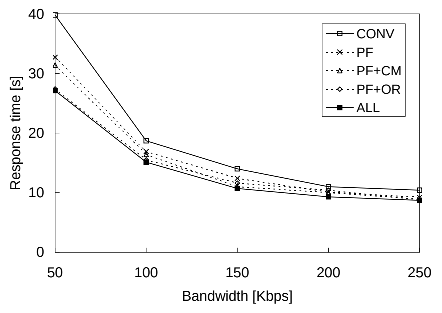
Figure 11(b) shows the performance differences between conventional web-browsers and proposed ones when the total number of objects increases. Two trends are shown in the figure. As more objects are included in a web-page, first, larger response time is expected; second, the response time reduced by the proposed solution is larger since all of the three mechanisms can gain more benefits.

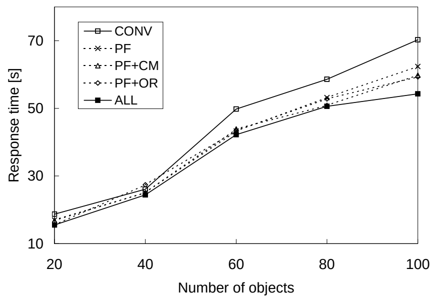
C. Impact of Number of Connections and Servers

Figure 12(a) shows the impact of number of connections to a single server, and it can be seen that up to 20% of response time can be reduced by using our solution. In the figure, as

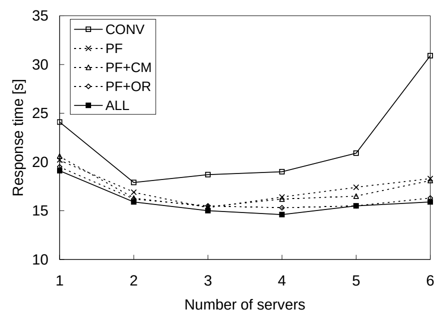




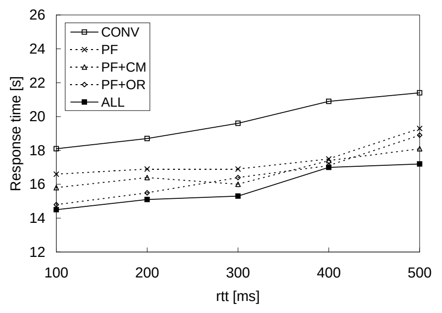




(b) Impact of number of objects



(b) Impact of number of servers



(b) Impact of round-trip time

Fig. 11. Impact of object characteristics

Fig. 12. Impact of numbers of connections and servers

Fig. 13. Impact of network characteristics

the number of connections to a single server increases, both the conventional and our solution has smaller response time. However, as this number exceeds 4, there are no obvious performance improvements with more connections. This result is consistent with the results presented in other works [13].

Figure 12(b) shows how the initial response time varies as the number of servers for a web-page increases. Two observations can be made from the figure. Increasing number of servers does not necessarily always result in better performance for both conventional browsers and proposed ones. Second, with more servers, our solution can achieve more improvements compared to conventional web-browsers.

D. Impact of Network Characteristics

Figure 13(a) shows how the initial response time changes under varying bottleneck bandwidth. As shown in the figure, our solution brings more performance improvement for smaller bandwidth. It is because of the fact that smaller bandwidth makes the screen contention problem identified in Section II more severe, and thus our solution can reduce response time more by alleviating this problem.

Figure 13(b) shows how the initial response time is affected by the rtt values. Since the major effects of rtt come from the request-response behavior of HTTP protocols (i.e. each object is fetched upon the request from the web client, and thus takes at least one rtt to fetch one object) and our solution can alleviate this effect by removing some of these rtts required, our solution sees better performance. As shown in the figure, around 20% performance improvement is achieved by our solution under the rtt values considered in the evaluation.

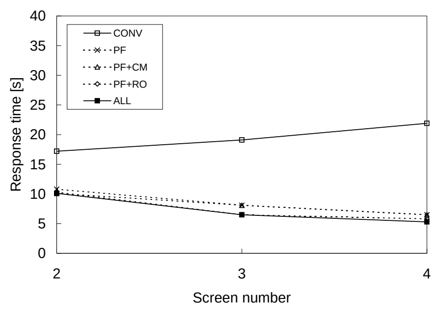


Fig. 14. Impact of fast scroll

E. Impact of Fast Scroll

Figure 14 shows the response time performance when a user performs fast scrolling. The x-axis of the graph shows the screen to which a user scrolls, and the y-axis is the response time. We assume that scrolling is performed when a web-browser completes downloading of the main HTML document and the entire document layout becomes available.

As seen from the figure, the response time increases when a user scrolls farther away from the initial screen for conventional web browsers. It is because conventional web-browsers perform greedy fetching without considering the locations of objects on a screen, and thus display of any screen requires downloading of all previous screens. In contrast, our solution has smaller response time as a user scrolls farther away from the initial screen. That is, if a user simply scrolls to the fourth screen, it can experience even smaller response time than any preceding screen! Since PF performs non-sequential fetching and fetches the current screen first, the response time does not depend on the screen number, instead, is determined by the *data size* in the current screen. Consequently, *as less data are located in farther screen (as seen in most popular web-pages), the response time for these screens is less than that for preceding screens*. Thus, we see a 70% reduced response time when the users jump to the fourth screen.

V. Related Works

In order to obtain optimization techniques related to web fetching, a lot of research such as [18] and [19] have studied the characteristics of embedded objects included in HTML documents. They have examined number, size, type, attribute, and file extension of web objects through millions of web-pages using their web tools or search engines.

To accelerate web browsing in today's Internet, especially for users who access Internet via low-bandwidth links, extensive research has been conducted and various approaches have been proposed. Besides the caching and server-side optimization techniques[20], most of these approaches require modifications on proxies or servers. For example, Gilbert at el.[21] proposed a new web delivery scheme that improves response time performance of images using progressive jpeg coding. [22] also proposed a distillation technique that controls the jpeg compression ratio in order to adapts to changing network environments. Some works suggest reducing image size via lossy compression, a lot of prototypes and commercial products such as UC Berkeley's Transend[23], Intel's Quick-Web, IBM's WebExpress[24], and Oracle's Portal-To-Go[25] have been developed. However, these solutions are difficult to deploy since it requires support from non-browser entities.

Web acceleration techniques such as [26] reduce user response time either through pre-fetching or predicted fetching on web-browsers. However, using these technologies requires either excessive bandwidth, thus degrades performance of other users or applications, or cannot guarantee 100% correctness. Compared to these techniques, our solution is free of these two concerns.

VI. Conclusions and Future Works

In this paper, we first explore the reasons why conventional web access models are not appropriate for low-bandwidth hosts. We identify two reasons, screen contentions and bandwidth under-utilization, which results in large user-perceived response time. To address this problem, we propose a new web access scheme for low-bandwidth hosts. The proposed scheme uses an intelligent mix of prioritized fetching, object reordering, and connection management. Using simulations with the web parameters obtained from the top web sites, we evaluate the performance of our scheme and prove its benefits over conventional web access models.

However, the performance evaluation through the simulations in this paper has not considered other factors that could possibly affect the user-performance in real communication networks, such as wireless packet loss, variance of roundtrip time, screen size in small devices, and so on. As a part of future works, we intend implementing a prototype of the proposed algorithms and investigating the impact of those other factors.

References

[1] Squid Web Proxy Cache, *http://www.squid-cache.org*
[2] IBM Websphere Edge Server, *http://www-306.ibm.com/software/webservers/edgeserver/*
[3] Sun Java System Web Proxy Server, *http://www.sun.com/webserver*
[4] Hypertext Transfer Protocol - HTTP/1.1, *http://www.w3.org/Protocols/rfc2616/rfc2616.html*
[5] J. C. Mogul, "The Case for Persistent-Connection HTTP," in *Computer Communication Review*, vol. 25, no. 4, pp. 299-313, Oct. 1995.
[6] M. Beck, D. Arnold, A. Bassi, F. Berman, H. Casanova, J. Dongarra, T. Moore, G. Obertelli, J. Plank, M. Swany, S. Vadhiyar, and R. Wolski, "Logistical computing and internetworking: Middlewarefor the use of storage in communication," in *Proc. of the 3rd Annual International Workshop on Active Middleware Services (AMS)*, San Francisco, CA, Aug. 2001.
[7] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari, "Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks," in *Proc. of the 11th WWW Conference*, Honolulu, Hawaii, May 2002.
[8] WAP FORUM, *http://www.wapforum.org*
[9] Binary Runtime Environment for Wireless, *http://brew.qualcomm.com/*
[10] Z. Wang and P. Cao, "Persistent Connection Behavior of Popular Browsers," in *Research Note*, December 1998. *http://www.cs.wisc.edu/cao/papers/persistent-connection.html*.
[11] comScore Media Metrix Top 50 Online Property Ranking, *http://www.comscore.com/press/release.asp?press=547*.
[12] The Network Simulator, *http://www.isi.edu/nsnam/ns*.
[13] R. Chakravorty, S. Banerjee, P. Rodriguez, J. Chesterfield, and I. Pratt, "Performance Optimizations for Wireless Wide-Area Networks: Comparative Study and Experimental Evaluation," in *Proc. of ACM Mobicom 2004*, Philadelphia, PA, Sep. 2004.
[14] Microsoft Internet Explorer, *http://www.microsoft.com/windows/products/winfamily/ie/*
[15] W3C Document Object Model, *www.w3.org/DOM/*.
[16] J. Pahdye and S. Floyd, "On Inferring TCP Behavior," in *Proc. of ACM SIGCOMM 2001*, San Diego, CA, 2001.
[17] A. Kuzmanovic and E. W. Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer," in *Proc. of IEEE Infocom 2003*, San Francisco, CA, Apr. 2003.
[18] T. Bray, "Measuring the web," in *Proc. of the Fifth International World Wide Web Conference*, Paris, France, May 1996.
[19] A. Woodruff, P. M. Aoki, E. Brewer, P. Gauthier, and L. A. Rowe, "An investigation of documents from the world wide web," in *Proc. of the Fifth International World Wide Web Conference*, Paris, France, May 1996.
[20] A. Datta, K. Dutta, H. M. Thomas, D. E. Vander, M. Suresha, and K. Ramamritham, "Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation," in *Proc. of 2002 ACM SIGMOD Conference*, Madison, WI, Jun. 2002.
[21] J Gilbert, and R. Brodersen, "Globally progressive interactive Web delivery," in *Proc. of IEEE Infocom 1999*, New York, Mar. 1999.
[22] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R.Walker, "Application-aware adaptation for mobility," in *Proc. of the 16th ACM Symposium on Operating Systems and Principles*, Saint-Malo, France, Oct. 1997.
[23] F. Armando, S. Gribble, Y. Chawathe, and E. Brewer, "The Transend Service," *http:// transend.cs.berkeley.edu/about*
[24] B. C. Housel, G. Samaras, and D. B. Lindquist, "WebExpress: a client/intercept based system for optimizing Web browsing in a wireless environment," in *Mobile Networks and Applications*, vol. 3, no. 4, pp. 419–432, 1999.
[25] Oracle Inc., "Oracal Portal-To-Go: Any Service to Any Device," in *Technical Report, Oracle Inc.*, Oct. 1999.
[26] Google Web Accelerator. *http://webaccelerator.google.com/*.