Cut-Load: Application-Unaware Content Partitioning for Web-based Information Access in Wireless Data Networks*

Tae-Young Chang, Aravind Velayutham, and Raghupathy Sivakumar
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332
Email: {key4078,vel,siva}@ece.gatech.edu

Abstract— **Web-based information access suffers tremendously in low-bandwidth wireless data networks due to the non-correlation between the content transferred across the wireless links and the actual data that is used to serve the user requests. As a result, the current web-access mechanisms face such problems as unnecessary bandwidth consumption, large response times, no service for partial disconnections, and low system utilization in wireless networks. In order to solve these problems with web-transfer in wireless networks, we present a new middleware for wireless web-access called *Cut-Load*, which performs application unaware content-partitioning in the graphical domain residing at both the mobile client and the proxy server that the mobile client communicates with. Cut-Load uses dynamic mode selection, opportunistic hoarding, transparent mode transfer, and display caching for efficient wireless web-access. Through simulations, we compare the performance of Cut-Load with that of the current web-access mechanisms and show that the proposed middleware brings significant performance benefits both in terms of bandwidth consumption and user-perceived response times.**

I. Introduction

Today, the majority of Internet users perform web-based information access using a web-browser. By continuous integration with various plug-ins, a web-browser has become an unified application to access not only HTML documents with attached files but also other web-friendly documents such as desktop publishing and presentation files[1].

As mobile computing technology has developed, users can browse these web documents on the Internet from their home, office, or elsewhere. However, in wireless environments, most network applications suffer from low bandwidth, large delays, and frequent disruptions in connectivity. These characteristics lead to several problems with current models of mobile web information access, such as excessive bandwidth consumption, large response delays, no service for partial disconnections, and inefficient system utilization.

The primary reason for these problems is the absence of flexible content partitioning in current file systems and the transfer of the entire content files from the backbone file server through low-bandwidth and less-reliable wireless links. In this context, graphical content partitioning is a concept that extracts and provides partial content that the user wants to view, and can be thought of as a What-You-See-Is-What-You-Fetch (WYSWYF) paradigm. Content-partitioning in the graphical domain can be realized without any dependence on applications. This is because any web-content can be represented using a common abstraction in the graphical domain without any relation to the nature of application using the content.

In this paper, we analyze how application-unaware content partitioning scheme in the graphical domain would solve the problems with traditional web-access systems. We also study the issues that arise with using simple content partitioning techniques instead of traditional binary-file transfer models. Briefly, the issues include larger bandwidth consumption for full-file access, no support for disconnected operations, and inefficient performance in the case highly compressed data. We then use the issues with a default graphical-domain content partitioning technique as the motivation to design and implement a new approach to mobile web information access.

We design and evaluate a new mobile middleware called *Cut-Load*[1], which uses application-unaware content partitioning along with the following unique design elements; (1) *Dynamic mode selection* solves the problem of inefficiency of content partitioning for certain types of content that is not conducive for graphical content representation such as highly compressed multimedia data; (2) *Opportunistic hoarding* helps in decoupling the response time that the user perceives from the fetch time of the entire binary file. This reduces the peak load of the system even while fetching binary content for satisfying future user requests; (3) *Transparent Mode transfer* allows the mobile client to switch the current access mode when the user accesses most of the content of a web-document in which case traditional binary-content transfer is more efficient than graphical content transfer.

The rest of the paper is organized as follows. Section II discusses the problems with traditional access models and substantiate the impact of the problems with real-life field experiments. Section III describes the notion of application-unaware content partitioning and the challenges of using it for mobile web-based information access. Section IV describes the details of the three elements and the operation of the framework as a whole. Section V evaluates the performance of Cut-Load with that of conventional web-access systems and discuss the impact of each design element in improving performance. Section VI discusses related work in the area. Finally, Section VII concludes the paper.

*This work funded in part by NSF grant CNS-0519733, and the Georgia Tech Broadband Institute (GTBI).

[1]The name is inspired from the Unix *cut* command, which "cuts" files into smaller parts

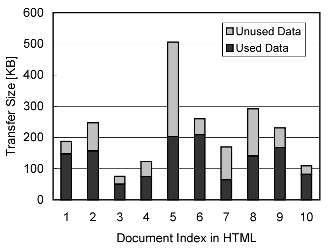

Fig. 1. Transfer Size in HTML

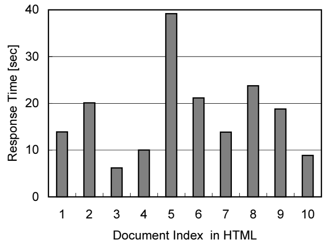

Fig. 2. Response Time in HTML

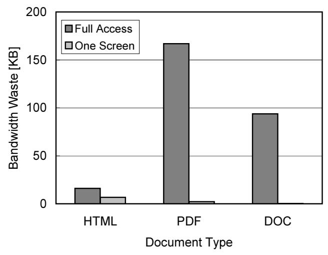

Fig. 3. Bandwidth Wastage

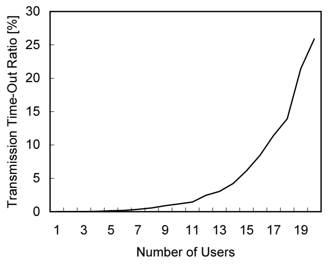

Fig. 4. Time-Out Rate

II. Motivation

In this section, we describe several drawbacks in the traditional model for mobile web access and use them as motivation for designing a new middleware for efficient wireless web-access. In order to measure the performance of traditional mobile web-access systems in low-bandwidth wireless networks, we use the CDMA2000-1X WWAN in 144 Kbps mode to access web-servers from a web-browser on HP N5430 laptop computer. We also use 10 web document files selected from the Top 50 Internet Sites and their links[2].

A. Not all content is always seen by users

Users generally don't view the entire content of a fetched file. In [3], it has been shown that 90% of users do not scroll down web pages but simply pick from the options that is visible on the initial screen when a page comes up. In spite of the fact that users may see only partial content that they are interested in, the conventional computing always consumes additional bandwidth to fetch unnecessary parts of content[2].

Figure 1 shows the total data transfer size for the entire raw data and the amount of data that the user ends up using for the 10 web document files. In order to get useful access sizes, we measure transfer sizes for the useful accesses using the percentage of each screen data with binary-size of content. We observe from the results that there is a significant difference in transfer sizes between full data transfer performed by current web-access systems and the necessary content.

B. Users suffer when response times are large

It is a well-known fact that there is a relationship between computer response time and users' perceptions. In [4], it is shown that users lose their concentration on their access when the response time is larger than 10 sec, and when it is over 1 minute, they lose interest and stop the current access. However, a mobile client in the traditional web-access system always waits until the entire content of a document is fetched regardless of which part a user wants to see. Fetching the unnecessary part of content increases the initial response time significantly and makes users impatient. Especially, in environments where available bandwidth is extremely limited such as WWAN, it results in extremely poor response time.

[2]Only some applications such as graphic viewers and add-on programs have limited capability for partial access to enable a user to access a fetched part of content before fully downloading it.

Figure 2 shows the response time results for the same set of the documents. In this figure, we observe that the time taken to download the web content is most often greater than the average user-tolerance limit, which researchers have observed in [4], [5], [6]. We note that download of only useful content can minimize transfer size, thereby response time without the degradation of content quality.

C. Larger file transmissions suffer from frequent disconnections

Wireless networks are prone to frequent disconnections due to attenuation of the wireless signal, fading of the wireless channel, interference due to other transmissions, and mobility of the client. This problem impacts the performance of traditional web-access systems in two ways namely: (1) The partially downloaded file until disconnection is generally not usable for serving the user requests for content, and hence the bandwidth expended to download the file goes waste; (2) The response time perceived by the user is increased because of the additional time spent in downloading the content again.

Figure 3 shows that the amount of bandwidth wastage is significantly smaller in the ideal case than in the traditional full file transfer case. This is because of the smaller probability of transmission failures caused by disconnections in the ideal transfer due to its smaller transfer time. It can also be observed from the results that the impact of frequent network disconnections can be alleviated by reducing the amount of content download using the wireless link.

D. Greedy transmission makes network utilization inefficient

We say that a connection has timed-out if the response time for the connection is greater than the specified latency tolerated by the user, and the user has stalled the download. When the connection is reset by the user, the bandwidth used in downloading the content till that point is wasted as mentioned earlier.

Figure 4 presents the percentage of connections that timeout as a function of the number of users in the network. In this figure, we observe that the time-out rates increase exponentially when the number of users (network load) is linearly increased. This exponential increase is due to the fact that the data downloads in traditional web-access systems use only greedy transmissions. Therefore, the higher peak load on the system degrades the system utilization and hence decreases the performance of the connections.

III. Application-Unaware Content Partitioning

In this section, we present the concept of graphical content partitioning for efficient web access and the issues with its use for wireless web-access.

A. *Graphical-domain content partitioning*

Since current file systems are *user-activity unaware*, they are not able to differentiate between the essential part of the file and the part that will not be used by the application. As a result, a file requested by an application is retrieved in its entirety from the backbone server irrespective of whether its content is eventually viewed by the user. Intuitively, this can be solved by partial access, and in reality there exist fetch-on-demand versions of some specific applications. However, it is infeasible to develop a generic application-unaware data partitioning technique.

Thus the only solution to application unaware content-partitioning is at the graphical level, i.e. content-partitioning at the output device level. Using this mechanism, content is abstracted in terms of the different inputs to the graphical user interface of the application. Hence, a web document is represented as a set of user-viewable screens that feed an output device. Note that this form of content-partitioning in the graphical domain can be performed in an application-independent manner because any web-content can be represented using a common abstraction in the graphical domain.

Now we describe how graphical-domain content partitioning solves the problems with traditional web-access model. We also present certain issues that arise due to the usage of pure graphical-domain content partitioning for mobile web-access.

- *Usage to fetch-size ratio:* Content partitioning in the graphical level is highly efficient especially when the byte-size of the entire content is large compared to the amount of the file that the user actually views, which is most often the case. However, for highly compressed multimedia content, it may not be efficient in terms of transfer size because of the performance limitation of re-compression in real-time . Hence graphical content-partitioning mechanism should be used selectively along with traditional full binary-content transfer techniques.
- *Response time:* Content partitioning enables quick fetch of user-accessed graphical content because of the smaller byte-size of initial graphical content compared to the entire binary-content. This helps in reducing user-perceived response time as the user requests for web-documents are served faster. However, when the user accesses all the content of the web-document, full binary-file transmission is better than graphical content transfer because of the repeated overhead incurred by graphical content transfer for serving user requests. Thus the graphical partitioning access mechanism should be supplemented with the binary-content transfer so that the mobile client can use the binary-content if the user accesses more than a threshold amount of content from the web-document.
- *Partial download disconnections:* Due to the large transfer sizes in the traditional model, there is a high probability of disconnection during content transfer. This in turn leads to increased response time and wastage of bandwidth. Since content partitioning brings benefits of transfer size reduction by dividing the accessed content into several parts that can be transferred individually, the probability of network disconnections stalling data transfer is small in the case of content partitioning techniques. However, even small size partitions may suffer from transmission failures caused by long-scale disconnections. Therefore, re-usability of partially downloaded graphical content is essential to serve user requests during disconnections.
- *Greedy fetch problem:* Content partitioning decouples the part that is needed for the initial access from the other part that is not required immediately. Therefore, it can minimize the impact on other network traffic by reducing the initial transfer size. Reduction of the greedy fetching size decreases the peak load duration of the system and minimizes the effect on the performance of other applications in a multi-tasking environment. However, for future disconnected operations, a full binary file may need to be fetched. In order to minimize the effect of this non-urgent transmission, it is useful to differentiate the greedy transmission for the initial part from the non-greedy or low-priority transmission for the remaining part.

B. *Thin-client computing*

Among currently existing computing models, thin-client computing provides the required abstraction for application-unaware and user-activity aware content-partitioning mechanisms in the graphical domain[7]. It is an extreme form of proxying where the proxy performs all the tasks on behalf of the client, and the only communication between the proxy and the client is for dumping screen data (proxy to client) and conveying user input (client to proxy). Formally, thin-client computing involves the use of a simple terminal or processing device connected to powerful servers where applications and data are stored and processed. A transcoding system on the servers adapts video, images, audio, and text to the individual device capabilities. Any remote content access for web-content attempted by the client results in the content being fetched by the proxy. Thus the thin-client model allows the client to be aware of the user inputs and respond accordingly before the entire data can be retrieved from the server.

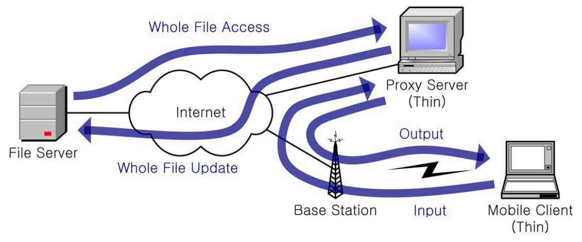

Fig. 5. Thin-client computing model

IV. Cut-Load Architecture

In this section, we propose a new mobile middleware that uses application-unaware content partitioning along with several unique design elements to provide an efficient web-access system for mobile clients.

A. Overview

We use the content-partitioning mechanism used by thin-client computing in the Cut-Load middleware. Cut-Load resides at both the client and the proxy as a middleware and hence it is easily deployable. The client-side middleware transfers a request of content access, manages content cache, and follows an application control message received from the proxy. The middleware at the proxy side decides the best computing mode for a requested content access, transfers objects for caching, and control applications at both the client and the proxy. Figure 6 presents an overview of the architecture.

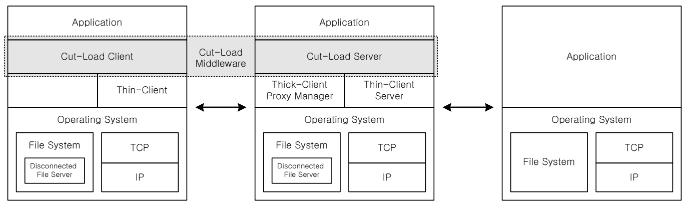

Fig. 6. Cut-Load architecture overview

Cut-Load operates in one of the following two modes.

- *Normal-mode:* In this mode, a Cut-Load client works in the same manner as a traditional web-access client. When a user wants to see content, the client fetches the original content files from the web-server and accesses them by running local applications. Because all required content files are fully transferred to the client, it can provide off-line (disconnected) operation with cached files.
- *Dual-mode:* A Cut-Load client works in dual-mode under specific conditions. In this mode, the client initially operates in thin-mode. While a user is seeing the content in the initial thin-mode, the client performs hoarding of original content files in the background [3]. When the content file is hoarded completely, the client notifies the user about the change of mode to thick-mode. Then, it opens the hoarded thick-data using an associated application and moves the current system focus to the application window. After it closes the remote application that the user used before, the mode transfer from thin to thick mode is complete.

In order to support these mode operations, Cut-Load consists of three basic elements, *dynamic mode selection*, *opportunistic hoarding*, and *transparent mode transfer* to address the issues with pure graphical content-partitioning mechanism. These elements bring benefits of faster access speed and efficient network bandwidth utilization.

[3]Because the hoarding is performed in an opportunistic non-greedy manner, we call it as *opportunistic hoarding*.

B. Dynamic Mode Selection

Thin-client content access is not always the clear winner in terms of performance[8]. Thus, in Cut-Load, the decision as to whether to use graphical content partitioning or not is done dynamically based on several factors such as thin-friendliness, size of content, and current network condition to maximize its performance.

For some types of content, the real-time compression processing in thin-client computing results in significant bandwidth inefficiency, and we call those *thin-unfriendly* content types. Usually, these content types are pre-compressed using a non-real-time algorithm, which requires spending a significantly long time to minimize their byte-size. However, de- and re-compression in real-time in thin-client computing may not reduce the content size enough, and this results in poor display performance with high body mass index (BMI) defined as the byte-size divided by the square of the pixel-size. Thus, thin-friendliness of content is decided by comparing the compression efficiencies in an off-line algorithm in thick-mode and a real-time algorithm in thin-mode. Besides the low re-compression efficiency problem, wrong selection of re-compression algorithm can be another problem. When a client accesses video content that is pre-compressed by a MPEG algorithm, the thin-proxy may not use a video compression algorithm, but a still image compression algorithm for re-compression. In this case, the re-compression process cannot be performed effectively, and finally this overload affects the server's overall performance for other clients.

When a user requests to access large pixel-size thin-unfriendly content, Cut-Load performs the dual-mode operation for faster initial access. However, an application itself generally does not provide pixel-size information, and it can be provided by a specific interface function to the application window manager, i.e. every application instance that has a window container needs to operate its own estimator. The estimator gets pixel-size information by measuring the scrollable size of the window container when a document is opened initially and when the current zooming rate in the application instance is varied.

In the mode decision process, the first factor that Cut-Load considers is the current network connectivity. Because interactive operations in thin-client computing are based on strong connection between a client and a server, the approach estimates the current signal-to-noise ratio (SNR) first to check eligibility for thin-client solution. If the current connection is not strong enough for thin-client computing, Cut-Load accesses the content in thick-mode regardless of other decision factors. Then, it measures byte-size, pixel-size, and thin-friendliness of content. The approach may access a thin-unfriendly document in thin-client mode to improve the initial access performance when the document has extremely large byte-size and pixel-size. However, optimal threshold values of those sizes are dependent on what a user accesses, therefore data mining of user access patterns is necessary in the approach.

C. Opportunistic Hoarding

By using a combination of both greedy thin-screen access and opportunistic thick-data access, dual-mode operation decouples the response time experienced by the user from the actual fetch time for the thick-content. This decoupling has two positive effects: (1) For users operating over a low-bandwidth link such as in a WWAN this significantly reduces the response time for large data sizes; (2) The decoupling facilitates a non-greedy approach to hoarding. This in turn reduces the peak hoarding rate and hence improves the system-wide utilization.

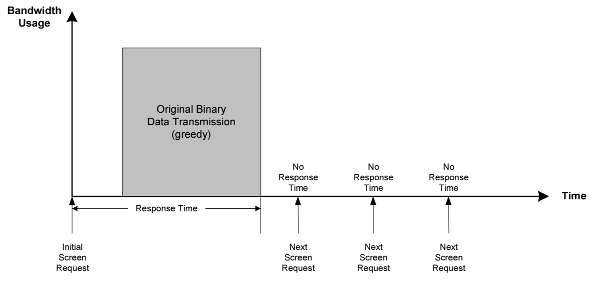

Fig. 7. Transmission in thick-mode

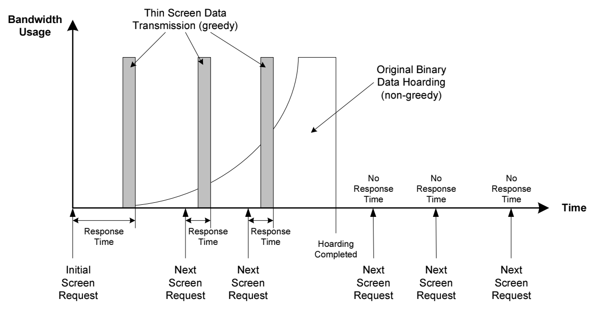

Fig. 8. Transmission in dual-mode

Opportunistic hoarding is performed when user requests for content are served using the thin-client mode of operation. However, unlike a normal content download that utilizes the entire available bandwidth in a greedy fashion, opportunistic hoarding is performed at an optimal adaptive data rate with lower priority to minimize the impact on the thin-client mode data transfers that use normal TCP with high priority. Thus, we use a rate control scheme called opportunistic hoarding rate control (OHRC).

In Cut-Load, the OHRC mechanism uses an exponential increase in the hoarding rate to perform opportunistic hoarding. It achieves a specific bandwidth share by using a weighted additive increase multiplicative decrease (W-AIMD) congestion control mechanism for performing opportunistic hoarding, i.e. it controls the weight to set a specific fraction of the bandwidth obtained by the high-priority thin-client flows that use normal TCP. When the hoarding starts, the flow is assigned a specific initial weight of 1/w, and this means that the flow would get 1/w of the capacity that a thin-client flow would achieve under the given network conditions. As the user performs input activity and the proxy server sends screen updates, the weight of the "hoarding" flow is increased by a pre-determined increment. After several increments the weight of the *hoarding* flow reaches 1, and from then on it would receive the same share of network bandwidth as a normal TCP flow.

This adaptive data-rate hoarding scheme has two advantages: (1) It avoids the starvation of *hoarding* flows so that after a sufficient time even in the presence of other normal TCP flows, the raw content would be hoarded and be ready for both efficient connected access and potential disconnected access.; (2) The exponential rate adaptation would ensure that sufficiently long *hoarding flows* achieve data rates comparable to normal TCP flows and hence complete the hoarding faster.

D. Mode Transfer

After opportunistic hoarding is completed, Cut-Load performs mode transfer to stop unnecessary bandwidth consumption. Our framework uses both thick-mode of operation as well as thin-mode for information access. When a client accesses content in dual-mode, it operates in thin-client mode initially. It also performs opportunistic hoarding of the raw data file in the background. If the client is still accessing the same content when downloading is complete, the framework changes the operating mode from thin-client to the thick client mode.

The transfer point of time is decided by the hoarding rate and the byte-size of hoarded data. When the mode transfer point of time is decided, the framework stops the current thin-client operation and notifies the mode transfer to the user. When the transfer is completed, it shows a message in a pop-up window and begins to provide access to the hoarded content file in normal thick mode. In order to provide a seamless user access after mode transfer, the environmental settings and system focus of both sessions in the client and the proxy should be synchronized.

Environmental settings are categorized into system settings and application settings. The system settings include screen resolution, keyboard layout, clipboard content, etc. The value of these settings are obtained by means of query messages to the operating system. The application settings include parameters set up within an application, such as menu bar, zoom rate, view option, etc. Because this information is application-specific and user-specific, the location of the application environment file should be input before it performs synchronization of application environment.

After the environmental synchronization is performed, the client synchronizes the various types of focuses. Mouse focus is the current location of the mouse cursor, and it can be located anywhere in the entire screen. Keyboard focus means the current position of the keyboard input, and it exists only when one or more text input controls are included in the current window. Screen focus is the screen position of the client area in the document layout. When the system is in thin-client mode and opportunistic hoarding is performed in the background, the client traces all the focuses. Once the new local application is opened, the captured focuses are restored by OS-specific interface functions.

V. Performance Evaluation

In this section, we evaluate the performance of Cut-Load and compare it with conventional web-access systems and discuss the impact of each design element in improving performance. In order to evaluate the performance and profile the benefits, we use Monte Carlo simulations.

A. Simulation Setup

The default screen resolution is 1024-by-768, and the pixel-size of the client area in the WebBrowser control is 1006-by-511. Each user access is simulated in terms of screen units, which have the same pixel-size as the client area. We use the average and standard deviation values of web-document byte-sizes from the experiments performed in Section II. We consider the case where the byte-sizes of document files follow single-side tail-less Gaussian distribution with a mean of 400 KB and a standard deviation 200 KB. The byte-sizes of each screen in thin-client computing follows a Gaussian distribution with the mean of 150 KB and the standard deviation of 50 KB. We assume the byte-size threshold value for mode selection to be 300 KB. Therefore, when a mobile client requests a thin-friendly document file larger than 300 KB, the client triggers the dual-mode operation automatically. In the WWAN environment, the overall cell capacity between the base station and the mobile hosts is assumed to be 640 Kbps. We use the IEEE 802.11 MAC protocol in the Point Coordination Function (PCF) mode for the WWAN model.

The internal data processing time is ignored, hence when a mobile client accesses already fetched or hoarded binary data, the effective response time is 0 second. All user accesses are performed by *PgDn* keystrokes in the unit of number of screens. The initial probability that a user accesses the next screen in the initial screen is 40%, and it is increased by 10% per next screen access up to 90%. With this probability model, each user sees an average of 1.468 screens per document. User access interval also follows a Gaussian distribution with a mean of 20 seconds and standard deviation of 10 seconds.

The access method for each requested screen is decided by the current access mode, current fetching status, and current hoarding status. If normal-mode is chosen, the initial screen access is provided only after the binary content file is fetched. After that, the other screen accesses in the same content are performed without any delay and any additional transfer. Therefore, the initial response time in normal-mode becomes relatively large, however once the file is fetched, additional accesses are provided instantaneously. If dual-mode is selected, the initial and all other screen accesses are performed in thin-client mode consuming bandwidth continuously until opportunistic hoarding is completed. After hoarding is completed and mode transfer is performed, all additional screen accesses in the same document are performed in normal-mode and do not require any delay and any more transfer as normal-mode. Therefore, the initial response time is relatively small, however this operation consumes bandwidth continuously until the hoarding is completed.

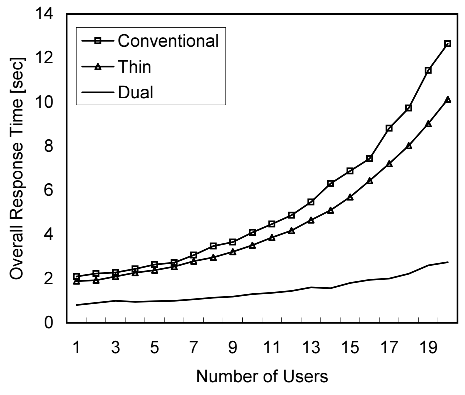

Fig. 9. Overall Response Time

B. Overall Response Time Performance

Figure 9 shows the overall screen response time using conventional thick-client model, thin-client, and a dual-mode client in Cut-Load. In the figure, regardless of the increased number of users, the dual-mode client shows better and relatively stable response time under 2 seconds. To the contrary, the thin-client and thick-clients show an exponentially increasing pattern when more users share the network bandwidth, and the thick-client shows somewhat worse performance than the thin-client.

This performance difference is explained by utilization of a judicious combination of thin-client and conventional computing models. When a user requests a large-size thin-friendly document, the dual-mode client selects the initial thin-mode to minimize the initial response time. As the user accesses more screens staying in the same content, the probability that the user will see the whole content increases, and then the dual-mode client increases the hoarding rate to reduce the future response time. Because the hoarding is performed in a non-greedy manner, it does not have a negative influence on other users' response time performance. In the figure, it is also seen that the dual-mode performance is not affected significantly by the number of users during the operation region. It means that the dual-mode consumes much less bandwidth in greedy transmission than other modes because only greedy traffic affects the overall response time performance.

C. Initial Response Time Performance

Figure 10 shows the initial response time results. In the figure, when the number of users is small, both the thin-client and the dual-mode client show similar performance, whereas the thick-client shows much worse performance because of its excessive initial overhead. As the number of users increase, the performance of both the thin-client and dual-mode client is not degraded significantly, and their initial response time is still below 10 seconds, which is generally accepted as the user tolerance limit [4]. However, the dual-mode client starts to show somewhat better performance than thin-mode operation.

The reason why the dual-mode client shows better initial response time performance than the thin-client even though the

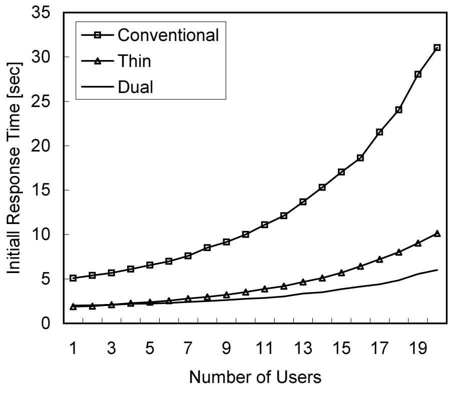

Fig. 10. Initial Response Time

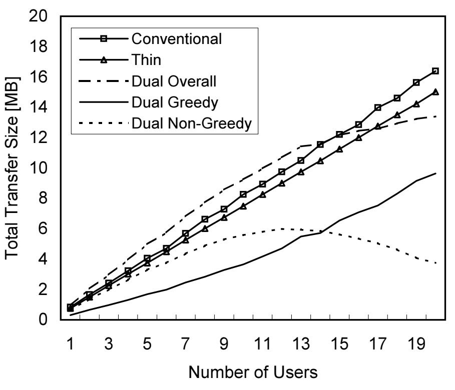

Fig. 11. Total Transfer Size

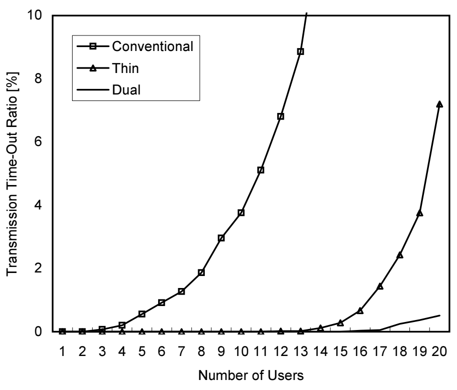

Fig. 12. Total Time-Out Rate

dual-mode client initially uses the same thin-client computing scheme is more efficient network bandwidth utilization. By the benefits of hoarding, other dual-mode clients don't spend bandwidth continuously as other thin-clients, hence the total greedy transmission size is relatively smaller.

D. Transfer Size Performance

Figure 11 shows the overall transfer size in the simulations. The performance in dual-mode is represented by the three curves of overall size, greedy transmission size, and non-greedy (opportunistic hoarding) transmission size. Its overall size is equal to the sum of the greedy and non-greedy size. In the figure, the conventional thick-client consumes slightly more than the thin-client. The overall transfer size of the dual-mode client is larger than or similar to those in other modes, however its greedy transfer size is only about a half of others. On the other hand, the non-greedy size begins to decrease when the number of users reaches 12.

It is generally true that a bandwidth-inefficient computing model consumes more bandwidth as well as shows worse response time performance. However, the dual-mode operations decouples the greedy transmission for the urgent part required immediately from the non-greedy transmission for the less urgent part of content. Therefore without disturbing other users' transmission it achieves significant response time performance improvement.

Another interesting point is that the non-greedy transmission has a peak point around a number of user of 23, and after this point the non-greedy size as well as overall size decreases. It means the available excess bandwidth for the dual-mode operations begins to decrease after the overall network utilization becomes saturated. At the same time, the overall transfer size begins to outrun because of the reduction of the non-greedy transfer size and the hoarding benefits that still exist. When the size of non-greedy transmission becomes almost zero due to the extremely heavy traffic, the transfer size overhead of the dual-mode becomes the same as that of the thin-mode.

E. Hoarding Performance

Because opportunistic hoarding mainly utilizes the excess network bandwidth available, as the excess available bandwidth is reduced by an increased number of active users, its performance is degraded. However, the increased background traffic also affects the main greedy transmissions, hence the dual-mode can keep dominance in the response time performance.

In Figure 12, we study the performance of opportunistic hoarding used in Cut-Load compared to conventional thick-client systems as well as pure non-greedy transfers. We had shown in Section II, that the traditional thick-client transfers suffer from lowered system utilization due to higher peak rates. We modeled system utilization as the transmission time-out rate ratio which is the ratio of the total number of connections which have response times greater than the average user tolerance level. It is shown in the figure that opportunistic hoarding used by our approach performs better than pure greedy transfers used by traditional file systems. This is due to the better system utilization because of the lower peak rates of opportunistic hoarding.

VI. Related Work

In [9], the authors propose a proxy that can transform data in new formats to old formats to accommodate thin-clients. Most software upgrades can then be performed at the proxy as opposed to at the client. However, the transcoding mechanism is not dealt with in detail since the paper presents it as one of many aspects of the proposed architecture. In [10], the authors propose a scheme to transparently support resource constrained mobile devices through powerful proxies. The proxy adapts its mechanisms to the dynamic nature of the wireless environment, and addresses the limitations of the client devices. The proxy provides filtering and compression of graphical images, converts postscript files to ASCII text, and does static data partitioning. Finally, in [11], the authors propose a combination of end-to-end and proxy-based approaches as an ideal solution for supporting mobile hosts. The proxy explicitly requests data from servers that has a resolution matching the present QoS and client capabilities.

In [12], the authors propose Spectra, a remote execution system in order to balance performance, energy conservation, and application quality. Even though it manages resources effectively in a mobile environment, it has a limitation of application dependency. Therefore, it needs newly structured applications for Spectra to work or modifications of current applications. In [13], the authors propose Puppeteer, a system for adapting component-based applications in mobile environment. Puppeteer has an advantage of adaptive transcoding execution by a proxy without modifying applications. However, it cannot overcome the quality degradation problem caused by limitations of transcoding in thick-client computing.

[14] is one of the first papers to analyze a real thin-client product, Microsoft Terminal Services. The authors show the performance analysis of CPU, memory, and bandwidth usage for several types of local applications. The focus of the paper is primarily on resource sharing in a multi-user environment, and the paper has limited analysis of the user behavior. [8] evaluates the web browsing performance of thin-client computing in a wireless environment. The authors focus on latency and size of pure transferred TCP data impacted by high packet loss rate over wireless network. However, their experiments are also executed in the simulated environment emulated by the wired network emulator, even though the network characteristics in a wireless environment are completely different from that in a wired environment.

VII. Conclusion

Traditional web-access systems are not tailored to perform well in low-bandwidth wireless networks. In this paper, we study the reasons why conventional web-server models are not optimal for wireless networks. We find that the reason for the inefficient performance is the operation in the binary level and being unaware of user activity. We evaluate the use of application-independent content-partitioning in the graphical domain as an alternative to binary-level file transfers for efficient web-performance for low-bandwidth wireless links.

We found several issues in using pure graphical content-partitioning techniques to serve mobile web-access requests. In addressing these issues, we propose and implement a new middleware for mobile web-access over wireless links. The proposed middleware uses an intelligent mix of binary file-transfers and graphical content-partitioning along with features such as opportunistic hoarding to reduce the bandwidth consumption as well as response times for web-access. We evaluated the performance of the Cut-Load middleware and proved its benefits over traditional web-access systems.

References

[1] List of browser plugins, *http://www.webdevelopersnotes.com/design/list_of_browser_plugins.php3.*

[2] comScore Media Metrix Top 50 Online Property Ranking, *http://www.comscore.com/press/release.asp?press=547.*

[3] Top Ten New Mistakes of Web Design, *http://www.useit.com/alertbox/990530.html.*

[4] R. B. Miller, "Response time in man-computer conversational transactions," in *Proceeding of the AFIPS Fall Joint Computer Conference*, 1968, vol. 33, pp. 267–277.

[5] B. Shneiderman, "Response time and display rate in human performance with computers," *Computing Surveys*, vol. 16, pp. 265–285, 1984.

[6] F. F. Nah, "A study on tolerable waiting time: How long are web users willing to wait?," in *Proceedings of the American Conference on Information Systems (AMCIS)*, 2003, pp. 2212–2222.

[7] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.

[8] S. J. Yang, J. Nieh, S. Krishnappa, A. Mohla, and M. Sajjadpour, "Web browsing performance of wireless thin-client computing," in *Proceeding of the 12th International World Wide Web Conference*, May 2003.

[9] Y. Chawathe, S. D. Gribble, T. Hodes, G. Nguyen, M. Stemm, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. N. Padmanabhan, E. A. Brewer, R. H. Katz, and S. Seshan, "A network architecture for heterogeneous mobile computing," *IEEE Personal Communications Magazine*, vol. 5, no. 5, pp. 8–24, Oct. 1998.

[10] T. Kunz and J. P. Black, "An architecture for adaptive mobile applications," in *Proceeding of the 11th Annual International Conference On Wireless Communication*, July 1999.

[11] A. Joshi, "On proxy agents, mobility, and web access," *Mobile Networks and Applications*, vol. 5, no. 4, pp. 233–241, Dec. 2000.

[12] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceeding of the 22nd International Conference on Distributed Computing Systems*, July 2002.

[13] E. Lara, D. S. Wallach, and W. Zwaenepoel, "Puppeteer: Component-based adaptation for mobile computing," in *Proceeding of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2001.

[14] A. Y. Wong and M. Seltzer, "Evaluating windows nt terminal server performance," in *Proceeding of the 3rd USENIX Windows NT Symposium*, July 1999.