BBN Technical Report No. 8340

Latency-Aware Information Access with User-Directed Fetch Behaviour for Weakly-Connected Mobile Wireless Clients

James P.G. Sterbenz, Tushar Saxena, and Rajesh Krishnan

9 May 2002

`http://www.ir.bbn.com/projects/wvm`

Prepared for:
Jean Scholtz and Gary Koob
DARPA Information Technology Office
3701 North Farifax Drive
Arlington, VA 22203-1714 USA

Prepared by:
Internetwork Research
BBN Technologies
10 Moulton St.
Cambridge, MA 02138-1191 USA

Latency-Aware Information Access with User-Directed Fetch Behaviour for Weakly-Connected Mobile Wireless Clients

James P.G. Sterbenz , Tushar Saxena, and Rajesh Krishnan

Abstract – **Mobile wireless clients have highly variable network connectivity and available bandwidth. There are times when they may be completely disconnected from the larger Internet. This dynamism of connectivity poses unique constraints for the problem of information access in general, and Web access in particular. These and other factors (such as loaded servers and congested networks) contribute to unpredictably high response times in content retrieval.**

We examine the problem of improving the utility of information access applications for these imperfectly connected devices. In particular, we are concerned with three related problems:

1. Providing information to the user on the estimated response time to retrieve content, the freshness of cached content, and the status on the strength of connection to the network. This information allows the user to make informed decisions about which content to retrieve, and how to retrieve it. We propose that client applications should be *latency-aware* with *location translucency*.

2. Giving the user control over how the client application behaves when desired content is either not fresh or absent from the cache, for example, whether the user waits for the content, uses an out-of-date cached version, or both. We propose that clients should support *user-directed* fetch behaviour.

3. Keeping the content cache as current as practical. In addition to conventional techniques such as demand prefetching and push preloading, we propose to build a profile of user activity. During periods of strong connectivity, the client application should *hoard* to keep the cache fresh, so that content will be locally available when disconnected from the network.

We describe the problem, through a motivating, application, and present our architecture, design, and prototype implementation, followed by a discussion of research issues to be explored. Our work draws on the experiences of designing mobile clients in other scenarios such as distributed file systems (e.g. Coda), as well the large body of work on Web caching and anticipation.

Index Terms– **high-speed, low-latency, mobile, wireless, weakly connected, disconnected, information access, web browsing, hoarding, caching, anticipation**

This work was sponsored by the Defense Advanced Research Projects Agency under contract N66001-97-D-8622 issued by SPAWAR.

James P.G. Sterbenz is with BBN Technologies in Cambridge MA, 02138-1191, USA, *jpgs@ieee.org*.

I. Introduction

Distributed information access applications generally provide *location transparency*, which provides a uniform view of information regardless of its location. Early examples of such systems include remote procedure call [4],[23],[17] and distributed virtual shared memory [24],[11]. The current canonical information access infrastructure is the World Wide Web, which provides a uniform hypertext linked view of content regardless of location. While location transparency provides useful information hiding to the user, sometimes too much is hidden. In particular, when the latency to retrieve data significantly exceeds reasonable interactive response times, the user may want or need to be involved in the process of deciding what data is worth waiting for. We argue that applications should therefore provide *location translucency* (e.g. [13]), and that client applications should be *latency-aware*. Latency information should be provided to the client application and, when appropriate, to the user. Furthermore, the user should be able to use this latency information to control the application client behaviour.

The rest of this paper is organized as follows: Section II presents the motivating application of distributed information access, and describes its characteristics. Section III describes the long latency environments we are targetting, in particular, mobile wireless networking. Section IV explains the concepts of latency-aware information access with user-directed fetch behaviour. Section V describes the architecture and implementation of our prototype client. Finally, Section VI presents issues for further research.

II. Motivating Application

Applications can be classified into three core classes—information access, telepresence, and distributed computing [18]—based on their relationship to one-another (client/server or peer-to-peer), bandwidth symmetry, transfer granularity, and end-to-end synchronization [26]:. In this work we are concerned with *distributed information access*, in which a *user* uses a *client* application to access content residing on a server at an arbitrary location. The information access application class characteristics are client/server, with asymmetric bandwidth requirements, potentially large transfer granularity, and no (or minimal) end-to-end

synchronization. The World Wide Web is the common infrastructure that realises this application class, and is the basis of our architecture and implementation. Note however, that the concepts described in this paper are not dependent on particular protocols (HTTP), clients (web browsers), or content formats (such as HTML).

In *interactive* information access applications, *response time* is the primary performance metric of interest to the user. Response time is the interval between the completion of the user action requesting content (e.g. mouse click or enter key) to the return of a usable increment of data (e.g. enough of a web page to fill the browser window). Human factors studies originally done in the context of time-sharing systems have long shown that user experience and productivity is highly dependent on response time, and that productivity is significantly higher when response time is subsecond [12]. As response time approaches the perceptually instantaneous (~100 ms), user behavior changes from *point-click-wait* to *point-click-point-click*. Visual pattern matching becomes more useful with less reliance on complex search queries. For example, when response times are subsecond, a user is more likely to scan through pages in a text editor than use a search expression, and more likely to click through multiple web pages than carefully construct a complex query to a search engine. The Web is essentially a distributed time-sharing system, to which these bounds should apply [25]. Furthermore, studies have shown that consistency and predictability in response time is critical to consistent user adaptation to response time.

Response time can be expressed as the sum of its components: $T_r = d_c + \mathrm{RTT} + d_s$. The client and server delays are d_c and d_s, respectively. The round trip time (RTT) through the network consists of the queuing delay at each hop, the speed of light delay along the path, and the transmission time based on the object size divided by the data rate b/r.

While it is possible to achieve subsecond response time in high-speed wireline terrestrial networks, approaching the 100 ms time is challenging for wide area networks. For example, the round trip time for a 20 000 km global WAN link is on the order of 200 ms. This round trip time exceeds a 100 ms latency budget, and uses a significant fraction of a 1 s budget. There are a number of network scenarios that make it significantly more difficult to engineer subsecond response, but for which we wish distributed information access applications to be usable; we will consider these in the next section.

There are a number of techniques that have been brought to bear to mask the impact of long latency on response time. These include caching that retains data on the assumption that it will be accessed again (e.g. [10],[29]), prefetching that anticipates data likely to be requested in the future (e.g. [1],[8],[16]), and presending content based on some knowledge located at the server of client preferences (e.g. server push preloading). Note that dynamic (or dark) content challenges all of these mechanisms (considered in Section VI.C).

While these techniques may significantly reduce the probability of a local cache miss, they do not *eliminate* it. New techniques are needed to help the user cope with unpredictably long response times for data that is not cached, or for which the cached copy is out-of-date.

We are concerned with improving user experience when desired content is *not* cached, in situations where the cost to the user is high in not being able to predict the response time. In environments that have the potential for response times that have high variance (from subsecond interactive through several-second "slow" to extremely long), the user can significantly benefit from *a priori* knowledge that allows intelligent decisions to be made before an attempt is made to access information. Currently, users attempt to access the information, and abort the request if it takes too long (*click-and-pray*). We propose to ameliorate this problem by providing response time estimates to the user, along with the ability to control the how desired content is retrieved.

Furthermore, in cases where a mobile client is episodically disconnected from the network, new techniques can be brought to bear that increase the likelihood that content will be locally cached when disconnected. Conventional anticipation techniques such as prefetching trade bandwidth for latency, which works well in high-bandwidth wireline networks. When connectivity is time varying, there will be periods of strong connectivity during which content should be hoarded, in anticipation of periods of weak or disconnectivity. The hoarding techniques we propose work in concert with conventional caching and anticipation.

III. Environment

We are concerned with any environment in which response times may be unpredictably long to the user. The extremely long latency links of the Interplanetary Internet [7] stimulated the original idea for latency-aware information access [26]. The motivating scenario for this research and prototype are mobile wireless clients in networks with weak and episodic connectivity. In this environment, it is the time varying bandwidth availability and intervals of disconnectivity that result in unpredictably long transmission delay components. Finally, wireline networks with significant congestion or loaded servers also provide an environment in which this work is useful. We will briefly describe each of these cases.

A. Very Long Latency Links

When latencies are very long, the penalty to the user is high in making a mistake in selecting which remote content to fetch. Consider the extreme case of the Interplanetary Internet, motivated by the NASA Mars missions. If a mission participant is using a web browser to access content, there is a significant difference in the time to access data located or cached within the Mars system from that required to retrieve data from Earth; in the latter case the round trip time is 8 to 40 minutes, depending on planetary alignment. Having feedback on the location of content or response time

estimates may determine which content is retrieved at a particular point in time.

B. Mobile Wireless Networks

Mobile wireless networks operate in an environment that is significantly more challenging than traditional wired networks. In wireless networks, multiple users contend for an open channel in a shared medium that is subject to interference and eavesdropping. Generally, the bandwidth available to users in current RF wireless networks is orders of magnitude below that possible in fiber optic networks[1].

Any bandwidth-limited network is response time challenged when content object sizes are large (b/r transmission time). For example, a 100 ms latency budget requires a T-3 link (45 Mb/s) to transfer typical web page images of 20–200 KB, and an OC-48 link (2.4 Gb/s) to transfer photographic resolution images [27],[26]. Furthermore, the available bandwidth in mobile wireless networks is not constant. Noise, interference, and (particularly in military scenarios) jamming contribute to time varying channel characteristics. Rain and snow can induce channel fades. As nodes move with respect to one-another, channel bandwidth varies as objects intervene in the communication path and multipath interference occurs.

Furthermore, mobility induces dynamic behavior on the topology of the network. As nodes move apart from one another, latencies can increase, and the network may be slow to adapt. All of these factors contribute to dynamic and unpredictable latencies of network paths, and the corresponding unpredictability of response times.

Finally, connectivity may be episodic, and there may be significant periods when the client is disconnected. In this case the response time includes the time to reconnect to the network.

C. Congested Networks and Loaded Servers

Wired terrestrial networks can also be responsible for long response times, particularly if there is a low bandwidth access dialup link, congested links that induce queuing in the switches, or loaded servers that do not have enough capacity to service requests at the rate received (long d_s). While it should generally be possible to engineer such networks to keep the latency budget subsecond, in practice this is not the case. Thus, techniques that help the user cope with unpredictably long response times [19] are also useful in the current terrestrial wired Internet.

IV. Latency-Aware Information Access Concepts

Currently, when Web browsing, no information is provided to the user along with the URL hyperlinks. Users click on URLs without any clue if the content is in the next room over a 100 Mb/s Ethernet link, or on a loaded server halfway around the world over low-bandwidth congested links. The only option is to click the URL and wait to see what happens, or doesn't happen in the case of an HTTP `404` response.

A. Latency-Aware Information Access

We propose to make the client *latency-aware* of the estimated response times and freshness of cached content, and to present these parameters to the user so that intelligent choices can be made as to what content to fetch.

We also propose to allow the user to exploit this information to determine what is fetched, and in what manner (described in Section IV.B). There are several types of feedback that should be presented to the user.

1) Response Time Estimates

Response time has a number of components, as indicated in Section II. Parameters necessary to compute response time include the end-to-end bandwidth (data rate) between the client and server r, latency along the path due to the speed-of-light delay and queueing in the network, size of the object requested b (ideally first usable increment), and delay on the server to service the request d_s. In cases where the client is (temporarily) disconnected from the network, an estimate on the time to reconnect is also necessary. These parameters can be determined or estimated by a combination of history (recent response time for the same object), meta-data (e.g. imbedding of object size in the URL), network monitoring information (e.g. SNMP MIB information on access links), and active measurements (e.g. probing to determine important end-to-end RTTs). Using this information, an estimate of the response time can be computed for content such as web pages.

2) Freshness Estimates

The other critical parameter about content is whether or not it is locally cached in (or near) the client, and if so, how fresh the cached copy is. A user is likely to be satisfied with a recently cached copy of a web page, but may not wish or be able to use an old copy. Thus, we need to present the freshness of information along with the response time estimate. A simple estimator is the age of the cached copy, but this does not reflect whether or not the cached copy is identical to the definitive copy. Server metadata embedding the last modified date in the URL of the definitive content is one way to provide a better estimate of freshness.

3) Connectivity Information

In mobile wireless networks from which the user may experience episodic connectivity, it is important to provide information to the user on the status and strength of the connection. If the user is connected, feedback can be presented on the bandwidth available and latency across the mobile wireless portion of the network. In the case of a wireless LAN, these parameters correspond to the access link to the base station. In the case of a vehicle with its own strongly connected network neighborhood (such as an airplane or ship), these parameters correspond to the wireless link connecting the vehicle with the fixed Internet. In some cases, the weakest link may be a satellite or fixed wireless link somewhere in the middle of the network, and it is

[1] Free space laser links will ameliorate the bandwidth restrictions, but line-of-sight requirements also result in connectively challenges.

possible that several links should be monitored for strength of connectivity.

If the client is disconnected, the disconnection should be conveyed to the user, along with an estimate of the time to reconnect. While estimating the reconnection time may be difficult, there are a number of scenarios in which a reasonable estimate can be made on past behaviour (e.g. disconnect from the wired net every morning at 8:00 to drive to work), known behaviour (trajectories of unpiloted aerial vehicles, spacecraft, and planetary bodies), or scheduled (planned meeting in a conference room that is opaque to RF). This connectivity information can be used in the response time estimates, and also to control profile-based hoarding (discussed in Section IV.C).

B. User-Directed Fetch Behaviour

While the response time estimates may help the user determine *which* content to fetch among a number of candidates, this is only part of the solution. Thus, we propose *user-directed* fetch behavior that provides explicit control on *how* a particular piece of content is fetched. The estimates on response time, freshness, and network connectivity allow the user can make intelligent choices on how to fetch particular content.

For example, if an out-of-date copy of a web page is in the local cache, and the response time estimate for the definitive version is long, the user can make a choice to:

- use the old cached copy;
- use the old cached copy while waiting for the definitive version; or
- wait for the definitive version to load.

Furthermore, with this knowledge, long-response time fetches can be non-blocking, that is Web browsing in the current window can continue, with a new window opened when the slow web page arrives.

C. Profile-Based Hoarding

There are a number of important techniques that mask latency for information access, such as caching and prefetching. Generally, these are not targetted toward the mobile wireless environment in which clients may be episodically connected or weakly connected to the wireline Internet. While using the established techniques, we also propose to *hoard* content while strongly connected [21], in anticipation of periods of weak or episodic connectivity.

A user profile can be constructed that tracks content access patterns, and is used to keep the local cache as current as possible by emulating user behaviour even when the user is not actively accessing content.

V. Prototype Implementation, and Operation

We have constructed a prototype latency-aware client with user controls on fetch behavior called WVM (Web Vade Mecum)[2]. The package consists of a set of client software that can be loaded on a Linux-based PC or mobile node. There are five main components to this client:

A. Modules that manipulate the request/response stream to present response time and freshness estimates
B. Network monitor to determine status and strength of connectivity of the client to the network
C. Modified browser providing user fetch controls and latency visualization preferences
D. Client cache
E. Profiling to construct and maintain a dynamic user behaviour graph and hoarding of content by traversing the graph

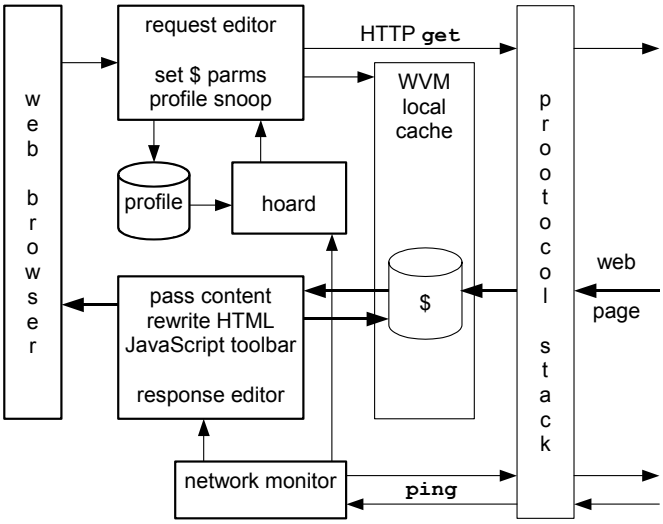

Figure 1. WVM Client Software Organisation

A block diagram of the WVM prototype is shown in Figure 1. Boxes with thick lines are modules specifically written for WVM or existing software that has been modified in this work (web browser). The operation of each of these components will now be described.

A. Client Response Time and Freshness Estimate

The WBI toolkit [28],[2] is used to trap and manipulate the HTTP requests and responses to/from the web browser. This has allowed the majority of our implementation to be done without the need to modify the web browser source. WVM modules written in Java manipulate the request and response stream as described in the rest of this section. The HTTP requests are observed by the *request editor* to build the profile graph (Section V.E) and direct the request to the WVM proxy cache, network, or both (Section V.C–D). The *response editor* manipulates the HTTP responses to provide response time, freshness, and connectivity information to the user. This is done by rewriting HTML and inserting JavaScript code. The original unmodified page is retained in the client cache to allow switching between display modes for each page. Note that the goal of this project has not been to design the ideal user-interface, but rather to provide an

[2] Available from http://www.ir.bbn.com/projects/wvm

interface that is usable for experimentation and further research.

An example WVM screenshot is shown in Figure 2. Two levels of information are presented. Coarse-grained information that is most likely to influence user behavior is presented by marking each URL hyperlink. Links are colored or tagged with icons[3] (or both) by having the response editor rewrite the HTML.

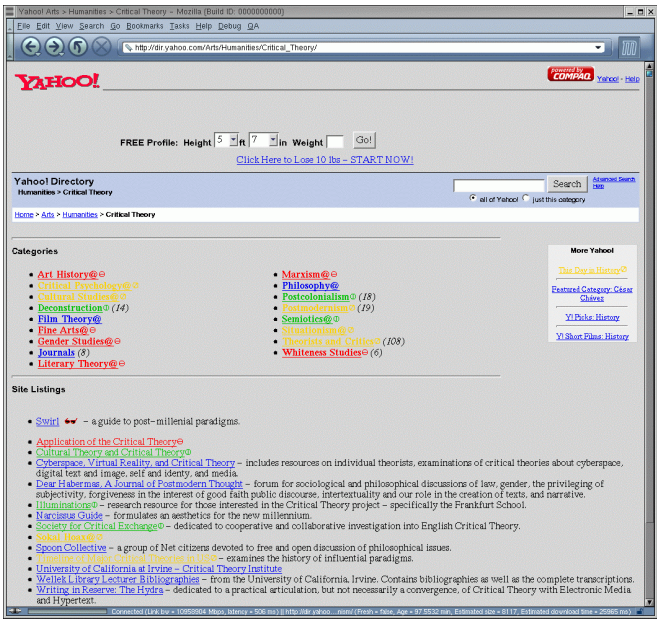

Figure 2. Web Page with Latency and Freshness Information

The type of tagging is user selectable by additional items in the View menu. A detailed view is shown in Figure 3.

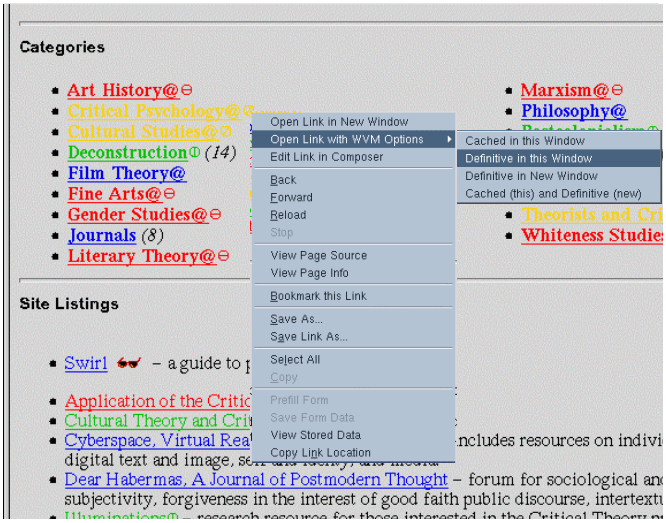

Figure 3. Tagged Hyperlink Details and Fetch Options

[3] The icons are derived from those used in railway signalling diagrams for the colors red, yellow, and green, which correspond to semaphore positions. This tagging is similar in concept to the traffic light WBI agent [3].

Conceptually, green links are fast, yellow links are fast (cached) but old (out-of-date), and red links are slow to retrieve. Links for which no estimate can be made are blue. A response time threshold $\hat{t}_r$ compared to the response time estimate t_i, and age threshold a_i are used to determine the link marking, as shown in Table 1.

Table 1. Link Marking

Color	Icon	Type	Latency and age
green	ⓘ	fast	$\hat{t}_r < t_i \vee \hat{a} < a_i$
yellow	∅	old cached	$\hat{t}_r > t_i \wedge \hat{a} > a_i$
red	⊖	slow	$\hat{t}_r > t_i \wedge \hat{a} = \infty$
blue	⊙	no estimate	$\hat{t}_r = ? \wedge \hat{a} = ?$

Note that for this purpose, precise estimates are *not* necessary. The user is interested in knowing whether or not clicking a link is likely to return a web page in reasonable interactive response bounds, rather than the precise time for the page to load. An order of magnitude of precision in the estimate is probably sufficient (e.g. 100 ms vs. 1 s vs. 10 s). It is more important to have a lightweight efficient estimator that is *recent*, rather than a complex estimate that is likely to be quickly out-of-date, and thus misleading.

In some cases this estimate may determine which link of a set of choices a user clicks; this is especially useful for search results and index lists, in which the user doesn't have a *particular* page that is needed. Response time estimates could be useful criteria for ordering search results. In the case where a *specific* page is needed, response time and freshness information are most likely to determine whether cached or definitive pages are requested, described in Section V.C below.

Detailed information, primarily for use in experimentations with the research prototype, is provided in a status bar at the bottom of the window, as shown by the example details in Figures 4 and 5. The response editor inserts JavaScript in the returning page to present this information. Two types of information are presented: per client connectivity, and per URL details.

1) Client Connectivity

On the left of the status bar, information is provided on whether or not the client currently has network access (Disconnected or Connected), and if connected the current bandwidth and latency, as shown in Figure 4.

Figure 4. Status Bar Details: Connectivity Information

These parameters are provided by the network connectivity monitor, described below in Section V.B.

2) Per URL Details

The right side of the status bar provides more detailed information about each URL on mouse-over, as shown in Figure 5.

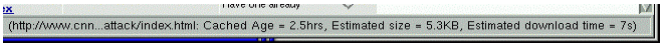

Figure 5. Status Bar Details: Response Time Estimates

The URL name is given, followed by the age if cached, estimated object size, and estimated response time. The estimated response time is based on history (previous fetches of the same object) modified by the current connectivity bandwidth and latency.

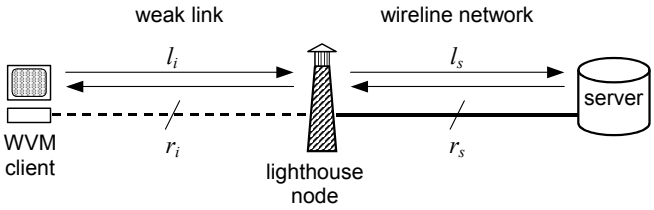

Figure 6. Strong and Weak Network Portions

Assume that a particular web page has been fetched i=1…n times, and the corresponding response times were $t_1 \ldots t_n$, the page sizes were $b_1 \ldots b_n$, the corresponding link bandwidths and latencies (see next section) were $r_1 \ldots r_n$, and $l_1 \ldots l_n$, respectively. Note that the bandwidth and latency parameters are from the client to the lighthouse, as described in the Section V.B. Let the bottleneck bandwidth and RTT from the lighthouse to the server be r_s and l_s respectively (see Figure 6). Since the most dynamic part of the path from the client to the web-server is expected to be the wireless link between the client and the access point (or the lighthouse), and since we are only interested in coarse estimates, we assume that the network conditions (r_s and l_s) from the lighthouse to the web-server remain relatively constant. Then the response time

$$t_i = l_i + l_s + \frac{b_i}{\min(r_i, r_s)}.$$

Therefore,

$$t_i - l_i = l_s + \frac{b_i}{\min(r_i, r_s)}.$$

Now, the response time $\hat{t}_r = \hat{t}_{n+1}$ can be estimated from past history as follows:

$$\begin{aligned} \hat{t}_{n+1} &= l_{n+1} + \text{estimate of}\left[l_s + \frac{b_{n+1}}{\min(r_{n+1}, r_s)} \right] \\ &= l_{n+1} + \frac{1}{n}\sum_{i=1}^{n}\left[l_i + \frac{b_i}{\min(r_i, r_s)} \right] \\ &= l_{n+1} + \frac{1}{n}\sum_{i=1}^{n}(t_i - l_i) \end{aligned}$$

Since all of t_i, l_i, and l_{n+1} are known quantities, the above formula can be used to estimate the current response time $\hat{t}_{n+1}$ for the next fetch of a particular page. The prototype does not currently use the bandwidth to the lighthouse and page size to modify the estimate; this should be done in conjunction with estimating the size of the first usable increment of the object, as described in Section VI.A.3.

Note that our prototype provides simple response time estimates that are more reliant on past history than we would like. Potential estimation techniques for further research are discussed in Section VI.A.

B. Network Connectivity Monitor

It is assumed that the location of the mobile wireless link or subnetwork is known that is likely to be weakly connected or disconnected. A *lighthouse* is positioned on the far side of this link, to determine baseline connectivity (connected or disconnected) and strength (bandwidth and latency) across the weakly connected wireless link to the wireline network, as shown in Figure 6.

In the case of a wireless LAN, the lighthouse can be the base station. In the case of a ship or plane, the lighthouse is on the other side of the wireless link to the Internet or proxy to the Internet. In the case of an ad hoc network, the lighthouse will be located at a connection between the ad hoc network and the Internet. While the lighthouse is statically configured in our prototype, we would expect it to be able to move periodically to reflect occasionally changing attachment points to the wireline Internet.

Bandwidth and round trip time estimates are determined by pinging the lighthouse with a set of small packets followed by a set of large packets. Periodically (every 30 seconds currently), the network connectivity monitor invokes `ping` with:

- destination = WIAB_LIGHTHOUSE
- number of attempts = 3
- timeout = 2 seconds

This is done for three 40 byte packets followed by three 540 byte packets. The average RTTs for the three attempts are stored as `rtt` and `rttbig`, respectively. If `ping` times out, the client is assumed to be in disconnected mode (`connected` is set to false, `rtt` and `rttbig` are set to large values `Long.MAX_VALUE` in Java).

The bandwidth estimate is then computed using the formula:

$$r = (540–40)\times 8 \,/\, (\texttt{rttbig}–\texttt{rtt})$$

Exponential weighted moving averages of r and l = `rtt` are maintained, and used to normalise the history based response time estimates described previously in Section V.A.

This strategy gives quick but coarse estimates (similar methods have been used in the past by others; a number of bandwidth measurement tools based on single packet and packet-pair strategies are available [5],[9]).

C. User Fetch Controls

Given response time and freshness estimates, the user can make an intelligent choice on how information is retrieved. This is accomplished by right-clicking the URL; we have modified the Mozilla [14] popup menus to provide a set of options that are input to the request editor.

Figure 3 shows this modified menu Open Link with WVM Options. Users can decide whether or not to use the

cached copy (Cached in this Window) or the original definitive version, either in the current window (Definitive in this Window) or non-blocking in a new window (Definitive in new Window). The non-blocking option is particularly useful when the response time estimate is very long. An option also allows opening both the cached copy in the current window, and the definitive version in a new window when it arrives (Cached (this) and Definitive (new)). This last option is useful if the response time estimate is extremely long for a needed definitive copy, but there is utility in using an out-of-date cached copy right away.

A further modification to the Mozilla View menu allows switching between URL tagging modes, as indicated above (color, icon, both color and icon, or original unmodified HTML). While this project does not intend to determine the best user interface for presenting response time estimates, we want WVM to be usable with a wide variety of existing web pages. Icon mode is intended for web pages that encode information in URL color, and provides accessibility for colorblind users.

Since the prototype HTML rewriting scripts cannot handle some fragile and complex content, original view allows using the page when the rewriting renders it unviewable.

D. *Client Caching*

WVM needs a client cache and can benefit from traditional techniques to maximise the probability of cache hits. For this we use squid [22] as a local caching server for the WVM client proxy. The squid cache on the local machine may operate independently, or be the lowest level in a network squid cache hierarchy. The browser cache must be disabled for WVM controls, feedback, and profiling to work properly.

The choice on the Open Link with WVM menu is used by the request editor to properly set HTTP fields used by squid to determine whether content is returned form the local cache, fetched from the network, or both.

E. *User Profiling and Emulation*

We have made one significant enhancement to traditional latency masking techniques that is targeted toward improving performance for disconnected and weakly connected operation. The profiler, which is a sub-module of the request editor tracks the HTTP requests and builds a *profile graph*; vertices are URLs visited and edges are transitions between them.

Figure 7 shows an OpenJGraph [15] WBI visualization of an example profile graph. Both vertices (pages) and edges (transitions) fade exponentially over time to reflect recent behavior and limit the cache size; this is shown by lighter color in the figure. They eventually fade out of existence if not requested by the user. Therefore, the profile graph is *dynamic* in the sense that not only does it change when the user accesses the web, but it also changes when the user is idle, and the profile graph vertices and edges decay and eventually expire.

For each page and transition, the prototype currently keeps track of the number of requests for each web page in the last {1 min, 5 min, 15 min, 1 hr, 1 day} time intervals. In addition, the latest time of access and the creation date is also recorded.

As mentioned earlier, WVM refreshes the local cache by emulating the user by walking this profile graph. The walk strategy is driven by the URL and transition statistics recorded during previous user requests. Currently, the strategy used by the prototype to refresh the cache is as follows. The latest access date is first used to prune the profile graph (a vertex or transition that has not been visited for an extended period of time is pruned). Then a breadth-first walk of the pruned graph is performed. At each level in the walk, the URLs at that level are scored based on the number of estimated hits to in the last one day. Then the URLs are refreshed in a descending order, based on this score (i.e. URLs with higher number of estimated hits are refreshed before those that have not been accessed as much in the last day). In the future, a more elaborate scheme using other statistics, such as the hit rate for smaller time intervals could be used to drive the cache refresh strategy.

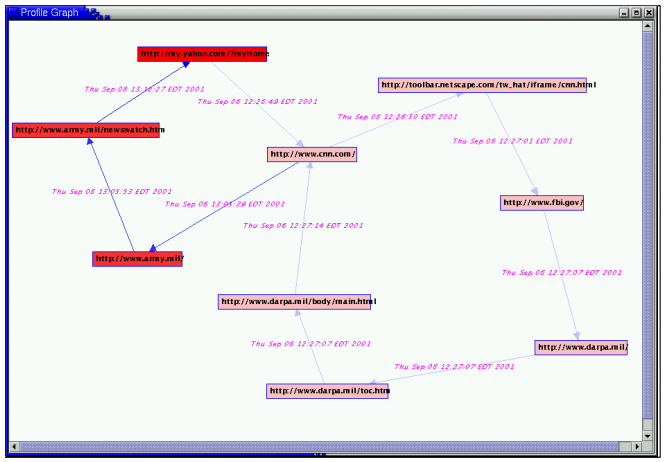

Figure 7. Profile Graph

To optimize disconnected and weakly connected operation we adapted the hoarding concept and state machine from CODA [21]. When strongly connected (high-bandwidth stable link), WVM hoards by aggressively walking the profile graph, fetching pages to keep the cache up to date. When multiple edges emanate from a vertex, the probability of traversing each edge is given by its weight based on time interval last traversed. When weakly connected, controlled hoarding should be done to carefully balance against user requests and scarce bandwidth in the shared medium (the prototype has hooks for this purpose). When disconnected, no hoarding can be done; all user requests must be served from cache.

In our prototype, we assume that the network can easily accommodate the additional load imposed by hoarding. A network based caching hierarchy, which pulls fresh content closer to the user, ameliorates the additional bandwidth

induced by hoarding. Future optimizations to the hoarding strategy are discussed in Section VI.B.

VI. Conclusions and Future Work

This goal of the WVM project has been to provide a framework for latency-aware information access, and to construct a prototype that can be the basis for further research.

We have designed an architecture that provides location translucency, so that client software and the user are aware of latency when it is important, and are able to control the manner in which information is retrieved. A network connectivity monitor allows response time estimates to be modified by the time-varying strength of mobile wireless links.

We have implemented a prototype, which is publicly available as open source, consisting of a set of WBI-based modules and a modified Mozilla browser. The WVM framework is modular, allowing individual components to be modified, enhanced, or completely replaced to experiment with new strategies.

While we believe that even this basic functionality based on simple estimates significantly enhances the current usability of the Web, there are a number of open questions to be answered, and further research to be done.

A. Response time estimates

Response time estimates do not need to be extremely precise, since they are intended only to guide the user in making choices between links and on whether to open cached or definitive copies of particular pages. A user probably doesn't care whether the response time will be 750 or 800 ms, but does care about the difference between 750 ms and 3 s. It is more important for the response time and freshness estimates to be reasonably *current*, and to estimate pages that have not been yet been retrieved (currently colored blue in the prototype). While the current prototype bases most of its estimation on previous fetches (modified by the current path characteristics to the lighthouse), there are a number of potential optimizations to consider for further work.

1) Network Connectivity Estimates

There a number of alternatives to the simple lighthouse strategy we employ in the protype. Some of these strategies could rely on more information from the network, for example on the strength of satellite links deep in the network. Some strategies could rely less on network information; the viability of purely client end-system strategies should be investigated.

2) Server Metadata

Since the server has current information about data that it serves, the inclusion of metadata in URLs could significantly improve the response time estimates. For example, if the server were to add a `size=nnn` parameter to the `<a>` hyperlink tag, the WVM client could use this as a current estimator to compute response time. If the page had been retrieved in the past, this estimator would be more accurate than the previous size; if the page had not been retrieved in the past, it could be used to estimate response time, which is not currently possible. Similarly, the server could add a `meta` parameter to the header to give a hint to its current delay in serving requests.

Additional improvements in the response time estimates could be gained by periodically probing web sites for which occasional but important requests are made [3]. For example, if the user occasionally requests pages from a news site, WVM could issue periodic HTTP `head` requests to determine the end-to-end latency and load of the server.

3) First Usable Increment

Currently, there is no mechanism to determine the response time of the *first usable increment* of a page. The response time that matters to the user is based on the time to fill the browser window, rather than the time to return the entire page. In the case of very long pages over low bandwidth links the response time would be is significantly over-estimated if the bandwidth to the lighthouse were used to compute transmission time . A crude modification based typical text page sizes might improve the estimate, but this is not sufficient for mixed content pages.

4) Mixed Pages with Inlined Images and Applets

A more difficult problem is how to make a reasonable estimate on the time to load pages with mixed content, such as inlined images, Java applets, and pages that use plugins such as Shockwave. In the case of progressively coded images, the estimate only needs to account for sufficient resolution that the image is initially useful to the user; server-based meta-data in conjunction with user preferences on desired resolution could assist in this estimate.

B. Scheduling

The current prototype does not schedule requests, other than serving current user requests before hoarding. There are three scheduling problems that should be considered in future work.

1) Aggressiveness of Hoarding

All anticipatory techniques trade bandwidth for latency. Excess available bandwidth is burned in the hopes that the latency for subsequent requests will be reduced. Clearly, this can significantly increase aggregate load on the network. In the case of WVM profile-based prefetching, it is important to understand the cost/benefit tradeoff, and when diminishing returns occur with increasingly aggressive hoarding.

Furthermore, significant gains are likely to be achieved with even simple optimizations based on past behaviour. For example, assume a user sleeps every night while the laptop is connected to the Internet over a broadband access link, and then disconnects for the commute to work. The benefits of scheduling hoarding mode for only the hour before typical disconnection time may offer the same benefits in cache currency, while using a order of magnitude less bandwidth.

Similarly, the way in which the profile graph is traversed warrants further research. A number of options can be considered in how to properly weight edges in the graph.

2) Current Requests vs. Hoarding and Prefetching

Generally, it would seem that current user requests should be served before prefetching and hoarding. It is not clear, however, that this is the proper long-term optimisation. For example, if individual pages are extremely long, and the user think time is short (period examining a partial page increment before the next mouse click), it may be better to reduce the rate at which current requests are served to allow future requests to be served with shorter response times.

3) Multiple Users

One of the scenarios motivating this work is a vehicle with multiple users, such as an airplane, ship, or train. In this case, the users share a LAN, and perhaps the WVM local proxy cache (the shared cache may be one level up the squid hierarchy). The scheduling problem in this case extends to multiple users. User requests must be scheduled against one another, as well as against requests to optimise long term performance described previously.

This problem relates to the standard operating system scheduling problems to provide quality of service or grade of service. For example, different fare classes on a commercial airline might receive corresponding grades of service; in a military scenario, the priority would be based on rank and mission requirements.

C. Dynamic Content

Dynamic (or dark) content is a significant challenge to virtually all latency masking schemes that rely on caching, prefetching, or preloading content. Dynamic content similarly challenges the WVM response time estimates and the benefits of profile-based hoarding.

Unfortunately, a significant number of web pages contain dynamic content with little or no thought on the performance implications. Web content should be divided into cacheable units, and the location of processing should be a conscious decision to optimise network bandwidth usage and user response time. It is better for a web page to have an embedded script or download a small applet than for the server to dynamically generate pages that cannot be cached, or in the case of WVM, estimated. WVM simply adds criteria that should be considered in the global optimisation of performance.

Similarly, dynamically generated web pages frequently have dynamically generated URLs. For example, a URL pointing to a top news story may change periodically. If the user is interested in whatever page corresponds to top news, rather than a particular archived news article, the dynamic URL obscures the ability to estimate response time or to properly hoard in advance. In this case, pages would be better to have stable aliases or links to the (changing) files. This is also a server optimisation that should consider overall long-term systemic performance.

D. Conclusion

We believe that WVM is a useful enhancement to Web browsing, giving users significantly more flexibility and control over distributed information access. WVM provides another step toward Vannevar Bush's vision of a *personal memex* [6], which locally maintains information on a mobile client that has been used in the past, as well as information likely to be needed in the future.

ACKNOWLEDGMENTS

The authors with to thank Jean Scholtz, formerly in DARPA ITO, for her interest and support of this project. Feedback from presentations at ETH Zürich, GMD Fokus, IBM Research Zürich, Technical University of Karlsruhe, University of Bern, UCI, University of Kentucky, and USC/ISI helped formulate this work. We also wish to thank Craig Partridge for useful comments and suggestions to improve this paper.

REFERENCES

[1] Alfred V. Aho, Peter J. Denning, and Jeffrey D. Ullman, "Principles of Optimal Page Replacement", *Journal of the ACM*, vol.18, no.1, ACM, New York, January 1971, pp. 80–93.

[2] Rob Barrett and Paul P. Maglio, "Intermediaries: An Approach to Manipulatin Information Streams", *IBM Systems Journal*, vol.38 no.4, 1999, pp. 629–641.

[3] Rob Barrett, Paul P. Maglio, and Daniel C. Kellem, "How to Personalize the Web", *Proceedings of ACM CHI'97*.

[4] Andrew D. Birrell and Bruce J. Nelson, "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, vol.2 #1, Feb. 1984, pp. 39–59.

[5] CAIDA, *Bandwidth Estimation: Issues and Approaches*, `http://www.caida.org/analysis/performance/bandwidth`.

[6] Vannever Bush, "As We May Think", *The Atlantic Monthly*, vol.176 #1, July 1945, pp. 101–108.

[7] Vinton G. Cerf, Scott C. Burleigh, Adrian J. Hooke, Leigh Torgerson, Robert C. Durst, Keith L. Scott, Eric J. Travis, and Howard S. Weiss, *Interplanetary Internet (IPN) Architectural Definition*, internet `draft draft-irtf-ipnrg-arch-00.txt`, work in progress, Feb. 2001.

[8] Edward G. Coffman Jr. and Peter J. Denning, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[9] James Curtis and Tony McGregor, *Review of Bandwidth Estimation Techniques*, New Zealand Computer Science Research Students' Conference, University of Canterbury NZ, 2001, available from `http://moat.nlanr.net/Papers/bandwidth-review.html`

[10] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz, "A Case for Caching File Objects Inside Internetworks", *Proceedings of ACM SIGCOMM'93*, (San Francisco), *Computer Communication Review*, vol.23 no.4, ACM, New York, September 1993, pp. 239–248.

[11] Gary S. Delp, Ronald S. Minnich, David J. Farber, Jonathan M. Smith, and Ming-Chit Tam, "Memory as a Network Abstraction, *IEEE Network*, vol.5 #4, IEEE, New York, July 1991, pp.34–41.

[12] Walter J. Doherty and Ahrvind J. Thadani, *The Economic Value of Rapid Response Time*, IBM, GE20-0752-00, 1982, available from `http://www.vm.ibm.com/devpages/jelliott/evrrt.html`.

[13] Maria R. Ebling, *Translucent Cache Management for Mobile Computing*, Ph.D Thesis CMU-CS-98-116, Carnegie Mellon University, Pittsburgh PA US, March 1998.

[14] Mozilla Web Browser, `http://www.mozilla.org`.

[15] OpenJGraph, http://openjgraph.sourceforge.net.
[16] Venkata N. Padmanabhan and Jeffrey C. Mogul, "Using Predictive Prefetching to Improve World Wide Web Latency", ACM SIGCOMM Computer Communication Review, vol.26, no.3, July 1996.
[17] Craig Partridge, *Late-Binding RPC: A Paradigm for Distributed Computation in a Gigabit Environment*, Ph.D. dissertation, Harvard University, 1992.
[18] Craig Partridge editor, *Report of the ARPA/NSF Workshop on Research in Gigabit Networking*, Washington DC, Jul. 1994, available from http://www.cise.nsf.gov/anir/giga/craig.txt.
[19] J.E. Pitkow and C.M. Kehoe, "Emerging Trends in the WWW User Population", *Communications of the ACM*, vol.39 no.6, 1996, pp. 106–108.
[20] S. Raghavan and Hector Garcia-Molina. "Crawling the Hidden Web", *Proceedings of VLDB 2001*, Sept. 2001.
[21] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment", *IEEE Transactions on Computers*, vol.39, no.4, April 1990.
[22] Squid, http://www.squid-cache.org.
[23] James W. Stamos and David K. Gifford, "Remote Evaluation", *ACM Transactions on Programming Languages and Systems*, Oct. 1990.
[24] James P.G. Sterbenz and Gurudatta M. Parulkar, "Axon: A High-Speed Communication Architecture for Distributed Applications", *Proceedings of IEEE INFOCOM'90*, IEEE, New York NY US, Vol.II, June 1990, pp. 484–492.
[25] James P.G. Sterbenz, "Protocols for High Speed Networks: Life After ATM?", *Protocols for High Speed Networks IV*, IFIP/IEEE PfHSN'94 (Vancouver BC CA), Aug. 1994, Gerald Neufeld and Mabo Ito, editors, Chapman & Hall, London UK / Kluwer Academic Publishers, Norwell MA US, 1995, pp. 3–18.
[26] James P.G. Sterbenz and Joseph D. Touch, High-Speed Networking: A Systematic Approach to High-Bandwidth Low-Latency Communication, John Wiley, New York NY US, 2001
[27] Joseph D. Touch, "High Performance Web", animation session, *Protocols for High-Speed Network*, IFIP/IEEE, PfHSN'96 (Sophia-Antipolis, FR), Oct. 1996.
[28] WBI: Web Intermediaries, http://www.almaden.ibm.com/cs/wbi.
[29] Duane Wessels and K. Claffy, "ICP and the Squid Web Cache", *IEEE Journal on Selected Areas in Communications*, vol.16 #3, IEEE, New York NY US, Apr. 1998, pp. 345–357.