Published in the proceedings of the 3rd USENIX Symposium on Internet Technologies USITS'01
San Francisco March 2001, pp13-24

Partial Prefetch for Faster Surfing in Composite Hypermedia

Javed I. Khan and Qingping Tao

Internetworking and Media Communications Research Laboratories
Department of Math & Computer Science
Kent State University
javed@kent.edu

Abstract

In this paper we present a prefetch technique, which incorporates a scheme similar to data streaming to minimize the response-lag. Unlike previous all or none techniques, we propose **partial prefetch** where the size of the lead segment is computed optimally so that only a minimum but sufficient amount of data is prefetched and buffered. The remaining segment is fetched if and only when the media is traversed. Thus, it delivers content without any increase in perceived response delay, and at the same time drastically minimizes unnecessary pre-load. The paper presents the scheme in the context of surfing in **composite multimedia** documents. It presents the technique and optimization scheme used for stream segmentation backed by analytical model and statistical simulation. We report remarkable increase in the responsiveness of web systems by a factor of 2-15 based on the specific situation.

Keywords: prefetch, streaming, multimedia.

1. Introduction

The success and failure of many web systems depend on the surfer's perception of the systems responsiveness. Caching and prefetching are the two principal techniques, known for improving responsiveness for systems that involve correlated data communication. Both the techniques have been extensively used in hardware systems to offset memory access latency. Caching techniques have been studied with much interest for the Internet. Also recently much focus has been shifted to prefetching. Some insight can be gained by a comparison between the two domains.

Despite the similarity, the caching in the Internet domain seems to be a harder problem. Hardware cache operates in a more predictable environment, where the parameters such as page size, and access times in the memory hierarchy are limited to few classes only. In comparison, the variability faced in the Internet is quite high. Apparently, the principle of locality is less conspicuous in the case of web. A number of recent studies, with various innovative caching schemes report caching efficacy in the range of 30-60% [4,7,10]. In comparison, a well-designed hardware cache can achieve a hit rate as high as 90% [1,18]. While several protocol issues are known to be involved in such performance degradation (such as validation requirement, and presence of Cookies and CGI scripts) but we also suspect one of the deeper reasons is the limited locality of reference. In the web, there is no short-term iterative construct such as 'while' of 'for' loops in the causal chain above. Consequently, it is not a complete surprise that the hit ratio is poorer here than in processor caches.

While, caching helps the case of repetitive references, prefetching reduces response lag for newer resources. It might be the case that prefetching may play much more important role in web than in hardware. A number of current studies, including ours, reported about another two times speedup with prefetching. Prefetching has also some additional complexity in the Web. Decision points mostly bifurcate the control flow tree in hardware due to the if-then construct. In contrast, the degree of branching in web is very high since there is no limit on the number of links in a page. In hardware quite often all the parallel branches are prefetched- and in some cases conditions can be pre-evaluated to determine the prefetch path. Neither is practical for web systems. We have also observed that prefetching often faces limitation due to excessive document transfer, which are never used later.

In this paper, we focus on this critical sub-problem-- ranking prefetchable date bytes. We discuss a technique, which provides optimum ranking of the prefetch nodes based on analytic considerations. In addition, we also show a technique where only a sub-part is fetched from the selected nodes. It cleverly uses the concept of *fragment streaming* to minimize the pre-load, without compromising the responsiveness gained by the prefetching in the first place. Each node is optimally divided into two parts-- the *lead* and the *stream* segments. During its operation, the system loads two parallel streams. In one, it fetches the stream segment of the current document, while in the other it prefetches the lead segment of candidate nodes.

Also we present the prefetch technique for an environment, which considers composite multimedia documents. An example is given is Fig-2 which shows a snapshot of a typical multimedia material from ABC News© website, serving combination of news summaries, images, voice integrated video clip as well as GIF animated banner advertisements from single link. Web pages with multiple embedded entities of various modalities such as applets, banners, audio, video clips, are becoming very common in power sites where responsiveness is critically valued. These are now typically composed of various media elements with varying playback requirements. In this paper, we also demonstrate how the size and mix of the lead segment can be computed optimally for such composite multimedia documents so that only a minimum but the right composition of individual media elements from these pages are prefetched and buffered. This problem has not been addressed before to such level.

2. Related Works

The Internet caching has been studied for quite some time [1,7,18]. It also has several implementations such as Harvest, Squid and CacheFlow [3,26]. The research in prefetching has gained momentum more recently [8,9,14]. In one of the pioneering studies, Kroeger et. al. demonstrated that with ample knowledge of future reference a combined caching and prefetching can reduce access latency as much as 60% [18,27]. The year after, Jacobson and Cao [14] proposed a prefetching method based on partial context matching for low bandwidth clients and proxies. Palpanas and Mendelzon [20] demonstrated that a k-order Markovian prediction engine similar to those used in image compression can improve response time by a factor of up to 2. Both these methods used variants of partial matching of context (past sequence of accessed references) for prediction of future web reference. These works suggested prefetching in-order of highest likely hood of access.

Pitkow and Pirolli [21] investigated various methods that have evolved to predict surfer's path from log traces such as session time, frequency of clicks, Levenshtein Distance analyses and compared the accuracy of various construction methods. This Markov model based study noted that although information is gained by studying longer paths, but conditional probability estimate, given the surf path, is more stable over time for shorter paths and can be estimated reliably with less data. Also of interest is the work by Cohen and Kaplan [4] who cited problems from bandwidth waste in prefetch, and as opposed to document prefetch suggested pre-staging only the communication session- such as pre-resolving DNS, pre-establishing of TCP connection and pre-warming

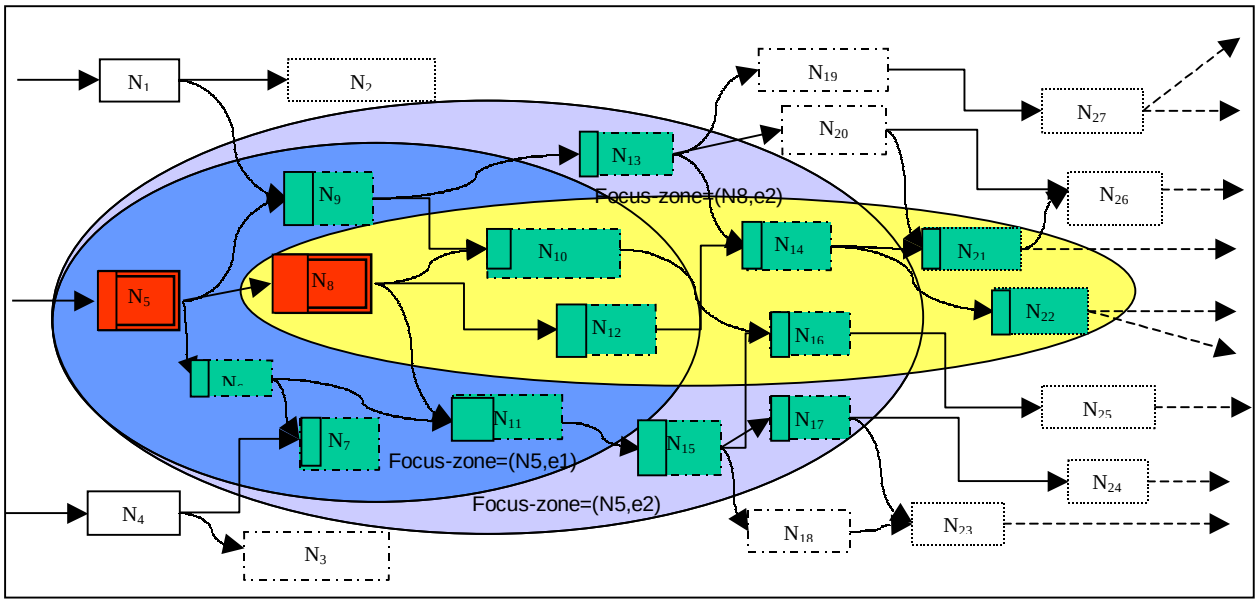

Fig-1: The hyperspace model used in the cache prefetching. The Focus-zones (N5,e1) and (N5,e2) show the roaming sphere for two pruning thresholds both anchored at node N5. The priority algorithm ranks the nodes 5-12 or 5-17 based on the selected threshold. The prefetch mechanism loads the nodes accordingly, while the reader is reading N5. By the time reader completes reading N8 is ready for rendering. When reader moves to N8, the roaming sphere is updated. The dotted box shows the actual media size, while the solid box represents the lead segments those are actually pre-loaded.

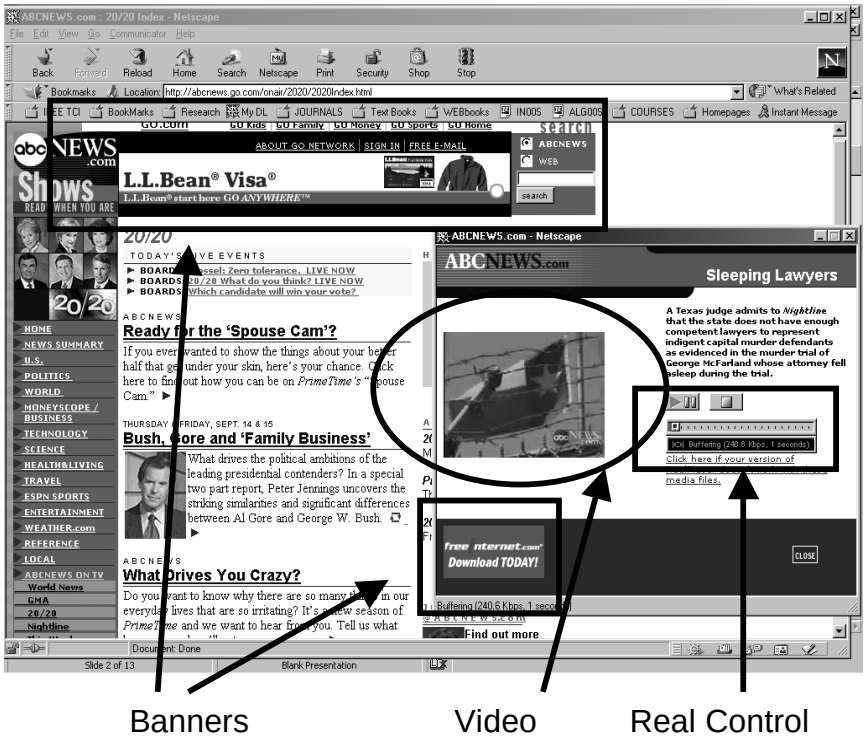

Fig-2 Example Composite Media from ABC News

Media Type	Files	KB
JPG images	5	18
GIF graphics	21	51
GIF animations		12
HTML	2	51
Video Stream	1	
Audio stream	1	
total	30	132
Table-1		

	Comp.	Audio	Video
Codec:		16Kbps Voice	RealVideo G2
minimum (Kbps)	86.8	~9	~77
maximum(Kbps)	123.4	~20	~102
Encoded/Clip BW	104.6	16	88.6
Average (Kbps):	121.7	18.7	92.8
Duration(min:sec)	3:12	3:12	3:12
Table-2			

Table 1 and 2 Static and Dynamic Properties of the Media

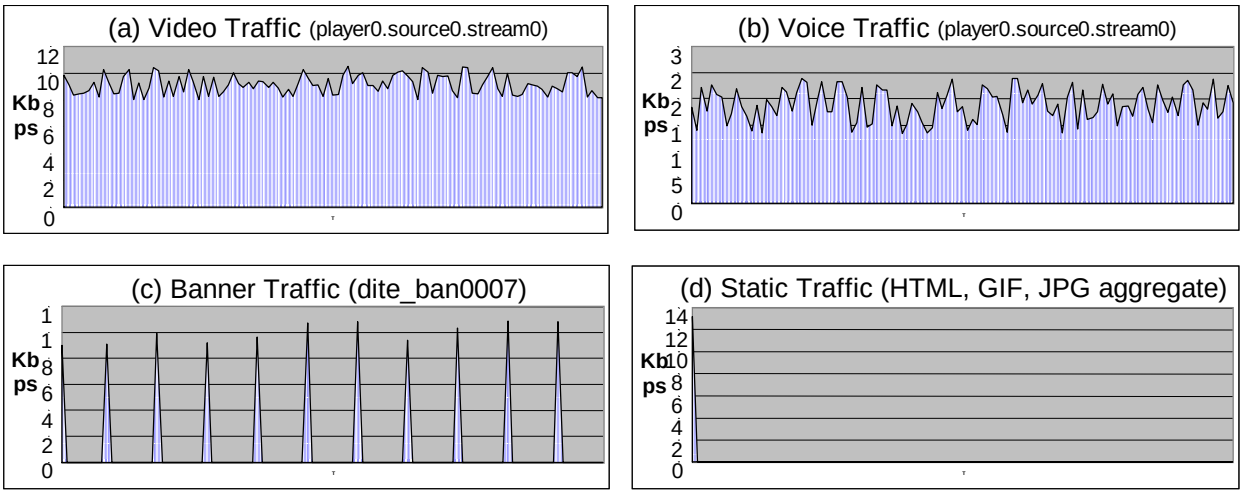

Fig- 3 shows the Rendering Rate Profiles respectively for the (a) Video (b) voice (c) Banner* (d) Graphics and Text and (d) composite media for the ABC News Nightline © Presentation.

by sending dummy HTTP HEAD request. RealPlayer (release 7) already pre-stages streaming connections linked from a page by pre-extracting and readying individual codec information associated with each.

In the context of the above works, we are investigating the issue of ranking-- particularly in view of the inevitability of prediction error. In a recent work we have shown ranking for low capacity links [16,17]. In this paper we extend the study in further depth—In addition to showing which nodes to fetch, we also analyze how much of each node to fetch, and for a composite document what should be the composition of the fetch segments. Instead of prefetching nodes simply in order of maximum probable transition paths, as used in most of the previous studies, we propose a ranking order which optimizes the response time with respect to all probable transition paths.

In this paper we further propose that instead of full documents we should only preload an estimated lead segment. The remaining can be loaded as a background streaming only when they have been requested. We show how to obtain the optimum lead segment that does not compromise the loading time- but reduces the amount of data loaded for documents that are never fetched.

The proposed method is based on linked transition probability and does not make any assumption about their means of estimation. It can be used with most of the methods proposed. Also to demonstrate its potential feasibility, in this paper we have sketched potential implementation technique only for the non-trivial steps. The actual deployment of prefetch system, including ours will require a much more detailed design work at the very protocol level. However, discussions about these issues cannot be accommodated in this paper. Rather, below we cite some additional interesting research, which we think will provide valuable insight about these aspects closely related to our work.

In related work, Duchamp [8] has discussed methods for clients and servers to exchange information for gathering and exchanging document usage statistics. Gruber, Rexford and Basso [12] have studied in detail the RTSP protocol extensions to support partial caching of lead segments of large multimedia documents. The concept of streaming for continuous media has been applied for quite a few years [15,22,23,24,25]. Grosso and Veillard have recently proposed XML extensions for document fragmentation [13]. Such fragment level communication can potentially be applied for streaming resources of other classes, as envisioned here. Crovella and Bradford [5] studied yet another advantage of prefetching- the reduction of burstiness of network traffic. Bangla et. al. proposed prefetching of only modified cached content to reduce cache communication in their proposal of 'optimistic-delta' [2]. Jung, Lee and Chon [15] reported error resilience of TCP prefetch-based long multimedia sessions on a long haul noisy channel as opposed to live UDP streaming.

In the following sections, we first present the model and the analytical considerations supporting the prefetch scheme. Section 4 then outlines key implementation issues. Finally, section 5 presents the performance of the scheme under various workload supported by rigorous statistical simulation.

3. Analysis

3.1 Hyperspace Model

First, we describe the model of the *hyperspace*. Fig-1 shows the model. Each *node* in the hyper-graph represents a web entity -- in the general case a composite multimedia document. Nodes are connected as per the embedded hyperlinks. The reader moves through a sequence of nodes in this hyperspace called *anchor sequence*. When a client (reader) program is active, the idea is to track the *anchor nodes*, monitor its neighboring regions, and prefetch selected parts from a subset of these nodes in client's *prefetch cache* with high likely-hood of traversal based on some optimization objective.

$H(V_H,E_H)$ denotes the entire hyperspace. $h(V_h,E_h)$ denotes the *visible sub-graph* of H about which the system has information. For tractability, it is further pruned before each optimization phase. We call this final sub-graph *roaming- sphere* and will denote it by $G(V_G,E_G)$. Although the node and link information of $h(V_h,E_h)$ or $G(V_G,E_G)$ is assumed available, but their content, however may not be resident in the prefetch cache at the beginning. Only $G(V_G,E_G)$ is used to determine the pre-loading schedule. A subset of its' nodes is eventually preloaded in the prefetch cache. $C(V_C, E_C)$ represents this final part of hyperspace, which is finally resident in the prefetch cache. Each node in $h(V_h,E_h)$ has node 'statistics' (such as size or load time). Also, each link in it has a transition probability p(i,j) associated with it. Links are bi-directional and transition probabilities are asymmetric.

3.2 Streamed Transport Model

Our transport model divides the transport into two probable phases. Each node thus has two transport parts-- the *lead segment* and the *stream segment*. The available bandwidth is correspondingly separated into two sub-channels; *feed channel* for loading the

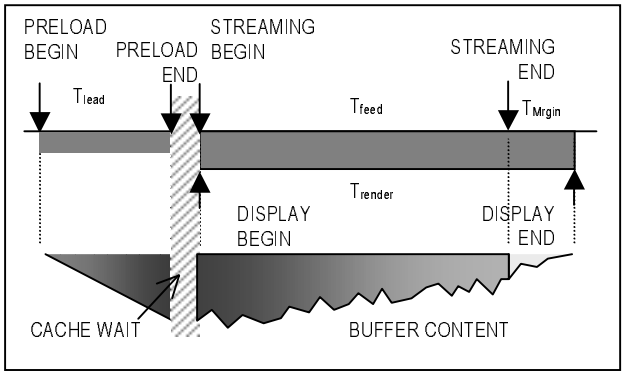

Fig-2 Streaming event model used for analysis. The top diagram shows event sequence and various time quantities and the bottom diagram shows the buffer occupancy under typical operation.

streaming segment of the current anchor, the *lead channel* to proactively load the lead segments of resources from the focus zone. Fig-4 shows the event sequence and an hypothetical buffer fullness for the transport model. The jaggedness of buffer fullness is due to the non-uniform nature of data consumption by the surfer. We assume that D_{total} is the size of an elementary resource, D_{lead} is the bytes in lead segment and D_{feed} is the bytes to be streamed. We use β to denote the ratio of total bandwidth to that allocated to the lead sub-channel. In Fig-1, the dotted box represents the media, while the solid head indicates the lead segments.

3.3 Composite Document Model

In this paper, we provide the entire technique in the context of composite document. A composite document has multiple embedded components with various rendering modalities. For example, Fig-2 shows a typical composite document with several embedded static as well as continuous media components (Tables -1 and 2 respectively provide a summary of the static and continuous media embedded here). To model such composite media we use a set of functions called *rendering rate profiles*-- one for each element. These profiles characterize the elements as a data rate function along the presentation time-line of the composite document. Note a profile is intrinsic to the document and is not dependent on the communication.

Given a composite document node N with embedded media elements n^i, we denote individual element's profile as the function $f^i(t)$, where superscript i is the element index. We use f(t) to denote the general variable rate profile. We also introduce the following three common profiles -- (a) constant rate (CBR) media, (b) impulse media, and (3) impulse series media.

Most CBR or audio and video plays generally are constant rate media. Parent HTML page, JAVA applets, or embedded foreground/background images generally contribute to the lead segment and is considered impulse media. A dynamic text or banner image requires cycles of bursts for presentation. Similar is the case with any large document (PDF, ps etc.) which are viewed page by page. Additional examples of this class of documents are stock ticks and charts, pushed banners, GIF animations as well as flash and slide shows. We use equation set-1 to model the profiles of these entities. Fig-5 sketches these functions.

$$\begin{aligned} f_c(t) &= R^i{}_c \\ f_I(t) &= H_I^i \\ f_S(t) &= \sum_{n=0} h_S^i \delta(t - nt_0^i) \end{aligned} \quad \ldots(1)$$

The document's *composite rate profile* is the sum of individual functions characterized by the coefficients R, H and h.

Given the above analytical model the problem that we will address in the remaining part of this paper is the ranking analysis at three levels. Given $G(V_G,E_G)$-- a hyper linked space of composite documents, and the associated node statistics (transition frequency *p(i,j)*, and node profiles f(t)*) we show* how to obtain the optimum prefetch sequence that will generate minimum expected response lag. Further, we will also determine the lead vs. feed segment sizes of the composite documents that will minimize the lag but with minimum waste of overall bandwidth. And finally, for the chosen composite documents, we also show how much of which components of the composite documents have to be brought in and when, for both its lead and streamed segments -- i.e. the *segment transport schedule* for the optimum performance.

3.4 Prefetch Node Ranking

The first question we address is what is the best prefetching sequence of the nodes that will minimize the expected cumulative *read-time lag* for a given network bandwidth? Below we present the results in the form of a series of proofs that ultimately will lead to the ranking algorithm. Proofs of these theorems are given in [16].

First we define the optimization criterion. In a hyper-graph G, Lets $U=(a_1,a_2,a_3\ldots.a_l)$, where $u_i \in G$, is the *anchor sequence*-- the sequence of nodes followed by a user. Let's Γ is the *loading sequence* in which the

nodes are loaded in the cache (Clearly, $U \subseteq \Gamma \subseteq$ {nodes in G}). Let p_i is the estimated probability that a user traverses a node n_i in roaming sphere G, and $T_{L,i}$ is the time the node a_i is fetched and $T_{P,i}$ is the time spent by the user in that node. Thus, we define an overall penalty function-- the expected cumulative *read-time lag*:

$$T(\Gamma \mid U) = \sum_{i}^{U} p_i \max\{[T_{L,i} - (T_{L,i-1} + T_{P,i-1})],0\} \quad ...(2)$$

The objective is to find the loading sequence Γ that will minimize the expected penalty E{T(Γ|U)}. It is important to note that this function optimizes with respect to all probable transitions of U, weighted by their transition probability. Given the above optimization criterion, it can be shown that:

Theorem-1 (Branch Decision)*: Let $A=n_c$ is the current anchor point with direct transition paths to a set of candidate nodes n_1, n_2, n_3,.. n_n, such that T_i is the estimated loading times of node n_i, and $Pr[a_{n+1}=n_i \mid a_n=A]$ is the conditional link transition probability, then the average delay is minimum if the links are prefetched in-order of the highest priority Q_i, where:*

$$Q_i = \frac{\Pr[a_{n+1} = n_i \mid a_n = A]}{T_i} \quad ...(3a)$$

This theorem states that at a simple branch point (roaming sphere of depth=1) immediately linked nodes should be prefetched in order of the conditional transition probability but in inverse order of their estimated load time.

The following theorem provides the relative priority between two nodes in a tree at different depths, which are not necessarily along the same path.

Theorem-2 (Tree Decision)*: If the sequence {n_1, n_2, n_3,...n_d} are the nodes in the path in a tree from the current anchor $A=n_0$ to a candidate node n_d, at depth d and T_d is it' s estimated loading time, then the priority Q_d is given by the product of the conditional transition probabilities along the path such that:*

$$Q_d = \frac{\prod_{i=c}^{d} \Pr[a_{i+1} = n_{i+1} \mid a_i = n_i]}{T_d} \quad ...(3b)$$

A corollary of this second theorem provides the relative priority of the nodes along a sequence. The result is quite intuitive.

Corollary-2.1 (Sequence Decision): *If two nodes are in a sequence, then the preceding node has to be loaded first.*

Unfortunately, the relative priority cannot be determined for a general graph for the optimization criterion defined by equation (2). However, under a slightly modified definition which ignores the credit due to cumulative read times along the paths ($T_{p,i-1}=0$ for all i in equation-2) it can be shown:

Theorem-3 (Graph Decision)*: For general graph $G(V_G,E_G)$ the node priority can be determined by computing order-n Markov state probability p_i. For a node n_i, with the estimated loading time T_i the priority function can be computed as:*

$$Q_i = \frac{p_i}{T_i} \quad ...(3c)$$

The above results indeed provide a close form solution to the node-ranking problem. Let's consider C is the current set of nodes in the candidate node queue. The sequence corollary ranks the nodes those are in a path. In a tree-structured roaming-graph, they effectively prunes the set C only among the immediate next nodes of the anchor thus, if we are at node 0, the candidate set can be initialized at $C_0 = \{children\ of\ n_o\}$, The Branch Decision theorem then ranks the nodes within this set. Let the winner be $n_{winner.}$ Then, when this is prefetched, it exposes its children nodes-- a new set of nodes, which now also becomes candidates for prefetch. Thus, the candidate set is updated to $C_i = C_{i-1} - \{n_{winner}\} + \{children\ of\ n_{winner}\}$. The Tree Decision theorem then provides the priroty of these new nodes compared to those which are already in the queue and thus completes the ranking. Finally, theorem-3 can be used in a graph for nodes which are child to multiple parents in C.

3.5 Critical Composite Lead Mass

The next question we address is how much of which node should we prefetch? This will also determine the pre-load time to be used in (2). Let us assume that the margin time=0. We use the constraint that the reading or rendering time should be at least equal to the streaming time for all components. First, we solve the simple case with only continuous media:

Continuous Media Case: Consequently, given a consumption rate R^i_{render} for the media type=*i* the amount of data that has to be prefetched is:

$$D^i_{lead} \geq D^i_{total}\left(1 - \frac{B^i_{feed}}{R^i_{render}}\right) = D^i_{total}\left(1 - \frac{\beta^i \cdot B_{channel}}{R^i_{render}}\right) \quad .(4)$$

Only the D_{lead} amount of data should be pre-loaded for mininum delay. It provides a lower bound and we call it *critical lead mass*. Corresponding pre-load time for the media is given by:

$$T_{lead} = \frac{\sum_i D^i{}_{lead}}{B_{preload}} \quad ...(5)$$

General case: The *critical lead mass* for a composite media can be determined by piece-wise integration of the combined rate function performed over a set of intervals defined as *negative zero crossing* (NZC) points where:

$$\sum f(t_i) - B_{feed} = 0 \text{ , and } \frac{d}{dt}\sum f(t_i) \le 0 . \quad ...(6a)$$

Fig-6 explains the segments where t_1, t_2 and t_3 are three such points dividing the presentation time T into four segments. The size of the required minimum lead segment is then given by the maximum of the piecewise integrals evaluated in 0-t_i:

$$D_{lead} \ge \max_{j=1}^{N}\left(\int_{\varepsilon}^{t_j+\varepsilon} \sum_i f^i(t)dt - B_{feed} \cdot t_j \right) \quad ...(6b)$$

For zero delay presentation, the system must preload equal or more. Note, if a segment sum negative in j^{th} interval, than it can reduce the piecewise lead segment of $(j+1)^{th}$ interval, however, the reverse is not true. The quantity ε is a small positive interval for ensuring inclusion of impulses in preceding intervals. Consequently, we look for maximum positive growth in the rate difference function (difference between the *composite rate profile function* and the *feed channel bandwidth),* however, this can be evaluated only by integrating at NZC points. For example, in the last segment of Fig-5(a) Since, the negative area A is larger than the positive area B , the integral from 0 to t_2, instead of from 0 to t_3 is the determining interval for its critical lead mass. Note equation-6 is a general solution including VBR rate profiles.

The profile set given by equation-1, however, allows faster computation. Accordingly, a composite document with continuous media rate R_c, an impulse media size H_I, and a impulse series media rate h_S per time unit t, for a presentation span of T sec, will require a critical lead mass of size:

$$D_{lead} = H + (R_c - B_{freed}) \cdot (T - t) + h\left\lceil \frac{T}{t} \right\rceil + \max\left[t \cdot (R_c - B_{feed}), 0\right] \quad ...(7)$$

3.6 Lead Segment Composition

Finally, we show the schedule of data segments and allocation of individual streams within D_{lead} and D_{feed}. In multiple parallel streams, excessive preload of one media can potentially result in sub-critical pre-load for the other.

Given the NZC points of the combined rate profile function, we segment the individual media entities into byte segments $B^i(t_j{:}t_{j+1})$, where i is the media index, and $t_j{:}t_{j+1}$ is the NGC intervals. We then define a *composite group* as $G_n(j)=\{B^i(t_j{:}t_{j+1})|$ for all i$\}$. A group contains bytes for all entities that belongs to the same jth NGC segment of the composite document node N_n. the following two rules then applies:

Rule 1: *To ensure critical pre-load all bytes in $G_n(j)$ should be loaded before the bytes in $G_n(j+1)$.*

Rule 2: *However, for lead prefetch, there is no requirement of ordering the bytes within a composite group $G_n(j)$, or between the groups from two nodes $G_n(j)$, and $G_m(j)$, when $n \neq m$.*

Consequently, within the same segment interval, the streams are ordered according to domain specific considerations such documents like stock ticks, can be left for fresh load, while any stored video can be loaded at lead segment.

4. System Model

4.1 System Description

Below we briefly describe the Proxy and Server mechanics for link statistics estimation, partial document fetch and bandwidth partitioning.

Link Statistics Collection: Statistics about links can be collected via client proxy that embeds link source in the HTTP 1.1 Request-Header Referrer field [11] each time when it issues a HTTP GET request. A server plug-in can track it and resolve the intra-server link references from the referrer id. A group of coop servers then can further resolve the remaining dangling links in batch mode by threshold driven periodic data exchange. The list of embedded link, profile parameters and access statistics for hot documents are can then be stored in a database called *hot-prefetch database* by the origin's stat server.

Statistics Propagation: There are quite a few strategy possible. A simple choice is to use a separate **GET_statistics** request method using HTTP reserve pool. Given an URL in this method the server plug-in can respond with the statistics stored in the *Hot-prefetch database*.

Partial prefetch: The conditional range GET mechanism of HTTP 1.1 (If-Range header, Range and Content-range, Response Code 206 Partial Content) can be used for specifying requests for media segments.

Bandwidth Partition Approximation: Bandwidth partitioning is not available in the current QoS-less Internet. Thus, instead a technique of byte proportioning per time segment basis can be used--

effectively creating the same result. Since all presentations are NZC segmented, first the Range GET requests have to be queued for each element from the current stream node based on its NZC time segments. Within each NZC segment, then the waiting Range GET requests have to be proportionately interleaved for the lead byte-segments earmarked for pre-load. The exact delivery time of the prefetch segments - as long as it is within the correct NZC segment, should not matter. Since, these do not require immediate presentation. However the feed requests are to be placed at the top of queue within each NGC segment. Thus, in effect stream data will enjoy a little prefetch.

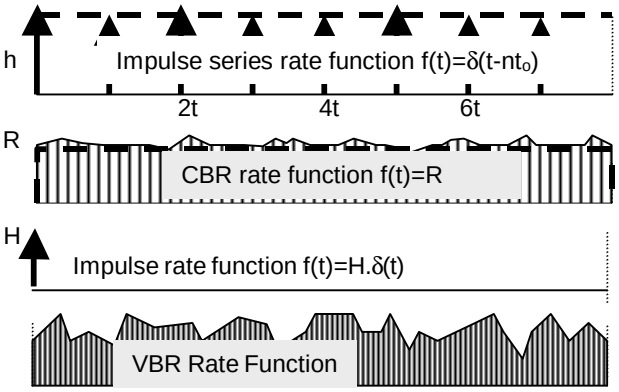

Fig-5 shows the profile for four typical traffic types given in equation set (5).

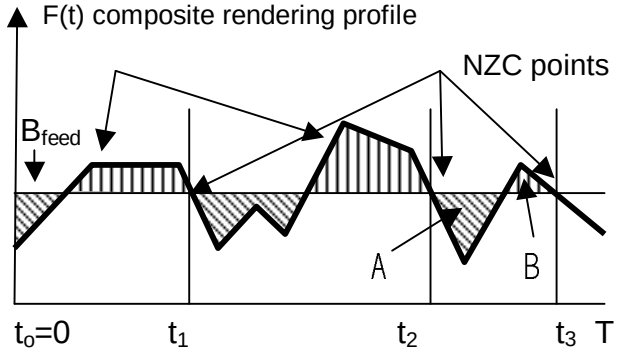

Fig-6 shows how the composite rendering profile can be rate for determining *critical lead mass*. There are three NZC points. In this example, the determining point is the middle one since mass A is larger than mass B.

4.2 Prefetch Mechanism

Now, we briefly describe the design of the prefetch scheme based on the above results. First, in the server side, the prefetch mechanism gathers the composite node statistics- the profile parameters and the access statistics compiled from previous references, in the hot prefetch database.

On the proxy side, when it detects a new *user-agent*, it initializes a new prefetch session for tracking its roaming sphere. The prefetch algorithm then computes $G(V_G,E_G)$ by recursive polling of the links and using a small cut-off threshold ε.

The prefetch mechanism then first computes the critical lead mass for each node on it. Based on this estimate it then determines the priority of the nodes according to the equation set-3(a)-(c).

It also orders the lead byte segments of the candidate nodes according to their group order using rule-1. Finally, the segments within the group are ordered according to rule-2 domain considerations. In our simulation, we rank the continuous media segments with higher priority and those of impulse series with the lowest to ensure late transmission of dynamic media. The selected and ordered prefetch segments are then queued in the fetch queue. The segments that are not in the lead mass however, may also be ordered according to the rules 1 and 2 but are placed on a separate dormant queue.

The current fetch queue is then merged with the feed queue that contains the segment schedule of the streaming segment of the current anchor node. The loading mechanism then generates the Range GET request accordingly.

The loading order remains valid until the current anchor node is read. Upon completion of reading node N, a new node is traversed, and the fetch and feed queues are reorganized. The subsequent evaluation is incremental. A new anchor point changes only the conditional probabilities.

The optimum ranking analysis presented here does not make any assumption about the prediction methods for estimating the link transition probability distributions and it should be applicable with any of the proposed methods [14,20,21]. These methods have varying degree of stability and prediction errors, and computational cost. However, for our simulation our design choice was the simplest one - the access frequency analysis based transition probability estimation. However, as can be seen later, we emphasized on testing the efficacy of the ranking procedure for varying prediction errors.

5. Simulation Results

To characterize the performance of the proposed scheme, we used statistically generated data set rather than server trace. We needed the composition and profile. Also we wanted to stress test the method under varying controlled conditions. Although, trace driven data provides detailed information about a

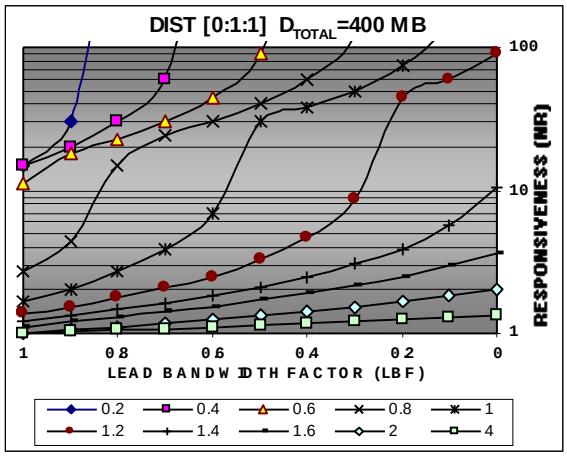

Fig 7(a)

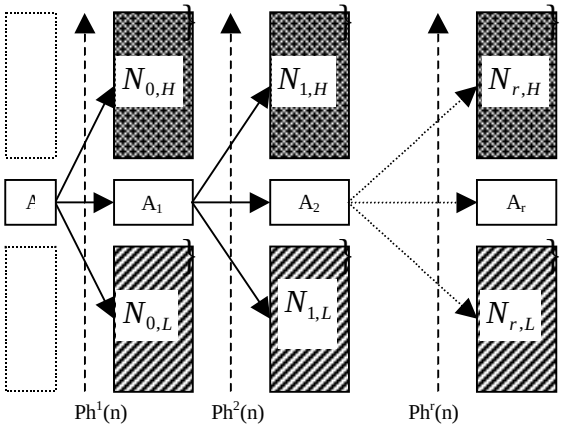

Fig-8 The ordered set of nodes at each state.

particular run, but the controllability of parameters is inadequate. Also the mix and distribution of web-page composition also seems to be changing significantly in short years. Consequently, for this particular work we found it more appropriate to generate broad range of hyperspace data with distinct and varying statistical properties and observe how the proposed method will perform in each of these conditions.

The objective of our first experiment is to observe how the introduction of partial prefetching -- particularly the relative bandwidth allocation between the fetch and feed channel effects. We were also curious to see how the relative rendering speed of various media types (such as text reading speed, play rate for audio or video) will impact the performance.

Consequently, in this first experiment we generated a random set of nodes (composite documents) each with a parent HTML impulse document (containing links to others) and a set of embedded CBR media. We limited the maximum links per node to 10. The underlying algorithm further pruned links with below ε low transition frequency and effectively considered only about 1-4 links for prefetch. HTML documents are given fixed sizes (H^i_S) . We generated the sizes for the CBR media and link transition probability using normal distribution. For a given link bandwidth we then varied the *rendering rate* (h^i_I) for the media as a control variable. Also, since our focus was to track the performance improvement only due to prefetching, we disbarred caching in the simulation. Thus, with each move to a new anchor, all nodes became prefetchable again. Fig-7 plots the observed reduction of the response delay for this experiment. It plots the *responsiveness* (lag-time with active prefetching normalized by that without prefetching). It shows the factor (y-axis) by which the lag time improves with respect to the percentage of bandwidth (x-axis) assigned to the fetch channel-- called *lead bandwidth factor* (LBF) We also plotted the ratio of *network bandwidth* to the *rendering rate*-- called *normalized rendering rate* (nRR). The curves show the performance when the *normalized rendering rate* varies from .2 (slow reading media) to 8 (fast playing media). As can be noted first, that with the inclusion of just prefetching (with 100% bandwidth assigned to the fetch channel or LBF=1.0), the read-time lag can be potentially decreased by a factor of 1-2 (shown by the left-most points) for media rendering rates 4-.8. However, as more bandwidth is set aside for streaming the improvement factor jumps. For example, it goes all the way up-to to 2-15+ times when 50% bandwidth issued for streaming. Not only that for media with slower rendering characteristics (such as text), the improvement is even sharper.

As indicated, the success of any prediction dependent algorithm depends on prediction accuracy. The next experiment we performed was to test how the proposed scheme fares against various levels of prediction discrepancies.

To model the problem, we let the user walk through a chain of anchor points, not necessarily always following the most probable transition path (the link with highest estimated transition probability or priority). However, we made the system to strictly following the ranking given by the equations 3(a)-(c) and 6(a)-(b). At the end of the run, for plotting, at each anchor point A_i, we grouped the nodes of the roaming-spheres into three groups. One with just one node-- the node which was actually traversed $\{a_{i+1}\}$(becoming the next anchor), the other with the nodes those received higher priority than the anchor-- node set $N_{i,H}$, and the third set with the nodes with lower priority than the anchor node--- set $N_{i,L}$. Fig-8 explains the grouping in for several successive anchor phases. We then aggregated the cumulative lead size of each set and observed the delay for various distributions of the bytes in these three sets. We denote the sizes respectively by $|N_{i,H}|$=H, $|N_{i,H}|$=L and $|a_i|$=A.

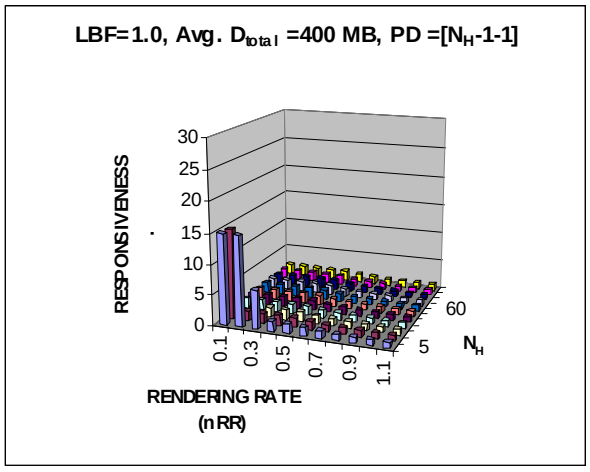

Fig 9(a)

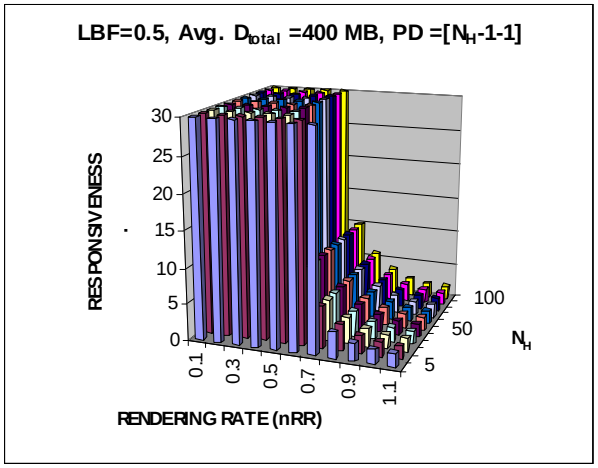

Fig 9(b)

(However, these are not necessarily the actual bytes prefetched, see next experiment for this). We modeled the error by a *prediction distribution ratio* H:A:L denoting the relative sizes of these three sets. We then conducted the previous experiment for various values of H, A and L. Fig-8 shows this grouping for each anchor stage.

First we present the performance for prefetch only model with entire bandwidth allocated to fetch (LBF=1.0). The 3D plot of Fig-9(a) shows the observed *responsiveness* (z-axis) both against the variation of *normalized rendering rate* (x-axis) and the *prediction distribution ratio* (y-axis), where the bytes prefetched before anchor (H) reaches all the way 100 times than the anchor bytes from zero, for the scheme with only prefetch (H had more impact than L). We observed, the system remains effective with high responsiveness (by a factor of +10) only when the rendering is slow (nRR less than 0.3) and the prediction error is small (H≈2).

We then repeated the experiment with partial-prefetch enabled. Fig-9(b) plots the same experiment with LBF=0.5. As can be seen the performance increases dramatically now. Not only the responsiveness improves (factor more than 15), it remained quite high for even relatively faster rendering resources (nRR around 0.7). It also remained immune to quite high level of prediction error (H>>2).

This dramatic result is not unexpected. The reason can be tracked by observing the background line loads. How much of the scheduled bytes (=H+A+L) are actually fetched depends on when the user moves to next anchor (Δ_{i+1}). If at this moment $N_{i,H}$, were loading then it would skip the remain and begin loading Δ_{i+1}. On the other hand, if it were in $N_{i,L}$, it would ignore its remaining part and render Δ_{i+1}. We define *load factor* as the ratio of the actual bytes prefetched to the bytes actually read (size anchor sequence). Fig-10(a) and (b) respectively trace this *load factor* (z-axis) for these same two cases (with and without partial prefetch). As can be noted that streaming enabled prefetching dramatically reduced the background line load--almost near one in large part of the graph.

The above results show how a streaming incorporated

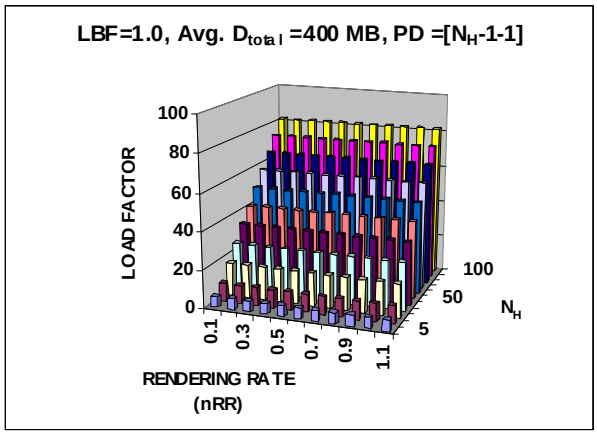

Fig 10(a)

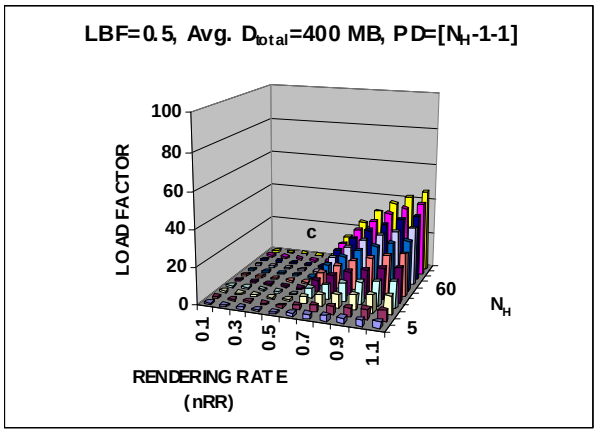

Fig 10(b)

active prefetching can significantly improve the responsiveness of a cache system. To summarize, the performance without partial prefetch, matched those reported by other researchers. However, the partial prefetch resulted in dramatic improvement. Careful reduction of data-bytes per node, in effect allowed information from more candidate nodes to be pre-fetched, given the same bandwidth and local storage.

6. Conclusions & Current Work

The prefetch prediction model reduces access lag for new references. It is quite appealing for the Web, and has attracted significant research interest. However, just prefetching seems to be limiting because of excessive transfer of unused bytes. This has been reported in few of the other research as well.

In this paper, we have suggested an innovative method that integrates a concept similar to streaming, and greatly reduces this waste, by prefetching only a carefully calculated amount. In this paper, we have outlined the method for a web composed of **composite documents**, which are becoming more ubiquitous in performance sensitive sites. We also presented the analytical considerations backing the design. Also, in this paper, we have presented simulation results based on statistical models that projects the scheme's performance under varying conditions. We think, such data optimization in prefetch is very critical because poorly done prefetch can adversely impact overall network performance.

The paper's principle focus is on the **data scheduling**. However, there are many other issues, which need further investigation. As indicated, the estimation of link transition probability, whether it is from conventional access log [21], or from explicit message exchange [8] as shown here, needs much more analysis.

In the systems design section we have only outlined implementation techniques for steps which appears non-trivial at first look- including link transition statistics collection or bandwidth partitioning on QoS less Internet. However, the actual deployment of any prefetch system, including ours will require more work on protocols. It seems that any prefetch will require prediction, which in turn will require exchange of document statistics between servers and server and clients. Also, the tracking ability of nearby hyper-graph, as demonstrated here will be useful. Also, interesting is protocol enhancements for partial document transfer for HTML, XML, RTSP entities [6]. Also, we did not present any prefetch algorithm, rather focused just on the ranking.

Within the scope of this paper, we deliberately switched off any caching. However, we are also currently studying the impact when caching is added, the results of which will be presented in a forthcoming submission. The points of interest are how the cache parameters --- cache size, media classification, and discard policies, interplays here.

As a part of our ongoing research, we are currently investigating an active net deployable prefetch proxy module, which can be dynamically launched as an active proxy inside network. The work is currently being funded by DARPA Research Grant F30602-99-1-0515 under its Active Network initiative.

7. References:

[1] Marc Abrams, Charles R. Standridge, G. Abdulla, A. Edward, Fox and S. Williams, Removal policies in network caches for World-Wide Web documents, ACM SIGCOMM , Stanford, CA, 1996, pp 293-305.

[2] G. Banga, F. Douglis, and M. Rabinovich. Optimistic Deltas for WWW Latency Reduction. Proc. USENIX Technical Conf., CA, January 1997, pp. 289-303

[3] High-Performance Web Caching White Paper, 1998 CacheFlow Inc. [Retrieved on June 8th, 2000 from URL http://www.cacheflow.com/technology/whitepapers/web.cfm]

[4] E. Cohen and H. Kaplan. Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency. Procs. of the IEEE INFOCOM 2000, Tel-Aviv, Israel, March 2000.

[5] M. Crovella, P. Barford, The Network Effects of prefetching, Proc. Of IEEE INFOCOM 1998, San Francisco, USA, 1998.

[6] A. Dan and D. Sitaram, Multimedia caching strategies for heterogeneous application and server environments, Multimedia Tools and Applications, vol. 4, pp.279-312, May 1997.

[7] F. Douglis, A. Feldman, B. Krisnamurty and J. Mogul, Rate of Change and Other Matrices: A Live Study of the World Wide Web, Proc. Of USENIX Symposium on Internet Technology and Systems, Barkeley, December 1997, pp-147-158.

[8] D. Duchamp. Prefetching Hyperlinks. Proceedings of the USENIX Symposium on

Internet Technologies and Systems, Colorado, USA, October 1999. [Http://www.usenix.org/events/usits99].

[9] Li Fan, Pei Cao, and Quinn Jacobson. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. Procs. of the ACM SIGMETRICS'99, Atlanta, Georgia, May 1999.

[10] A. Feldmann, R. Caceres, F. Douglis, G. Glass and M. Rabinovich, Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments, Proceedings of INFOCOM 99, 1999.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk & T. Berners-Lee, Hypertext Transfer Protocol HTTP/1.1, RFC 2068, January 1997.

[12] S. Gruber, J. Rexford, and A. Basso, Design considerations for an RTSP-based prefix caching proxy service for multimedia streams, Tech. Rep. 990907-01, AT&T Labs - Research, September 1999.

[13] Grosso, Paul, Daniel Veillard, XML Fragment Interchange, W3C Working Draft 1999 June 30, [Retrieved from: http://www.w3.org/1999/06/WD-xml-fragment-19990630.html]

[14] Q. Jacobson, Pei Cao, Potential and Limits of Web Prefetching Between Low-Bandwidth Clients and Proxies, 3rd International WWW Caching Workshop, Manchester, England, June 15-17 1998.

[15] J. Jung, D. Lee, and K. Chon, Proactive Web Caching with Cumulative Prefetching for Large Multimedia Data. Procs. of the 9th International World Wide Web Conference, Amsterdam, Netherlands, May 2000.

[16] Javed I. Khan, Ordering Prefetch in Trees, Sequences and Graphs, Technical Report 1999-12-03, Kent State University, [available at URL http://medianet.kent. edu/ technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/~javed/medianet]

[17] Javed I. Khan, Active Streaming in Transport Delay Minimization, Workshop on Scalable Web Services, Int. Conf. on Parallel Processing, Toronto, August 2000, pp95-102.

[18] T. Kroeger, D. D. E. Long & J. Mogul, Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, Proc. Of USENIX Symposium on Internet Technology and Systems, Monterey, December 1997, pp-319-328.

[19] NLANR, Proxy cache log traces, December 1999, ftp://ircache.nlanr.net/Traces/.

[20] T. Palpanas and A. Mendelzon,, Web Prefetching Using Partial Match Prediction, WWW Caching Workshop, San Diego, CA, March 1999

[21] P. Pirolli and J. E. Pitkow, Distributions of surfers' paths through the World Wide Web: Empirical characterizations, Jounral of World Wide Web, 1999, v.1-2, pp29-45

[22] J. Rexford, S. Sen, and A. Basso, A smoothing proxy service for variable-bit-rate streaming video, in Proc. Global Internet Symposium, December 1999.

[23] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A transport protocol for real-time applications,' RFC 1889, January 1996.

[24] H. Schulzrinne, A. Rao, & R. Lanphier, Real Time Streaming Protocol (RTSP), RFC: 2326 , April 1998 [Retrieved on June 8th, 2000 from URL ftp://ftp.isi.edu/in-notes/rfc2326.txt]

[25] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In Proceedings of the IEEE INFOCOM'99 Conference, 1999.

[26] Squid Web Proxy Cache ,http://www.squid-cache.org, 1999.

[27] Z. Wang and J. Crowcroft, Prefetching in the World Wide Web. in procs. of IEEE Global Internet, London, UK, 1996.