

Proxy-Cache Aware Object Bundling for Web Access Acceleration

Chi-Hung Chi, HongGuang Wang, William Ku
National University of Singapore
Email: chich@comp.nus.edu.sg

ABSTRACT -- In this paper, we address the acceleration of web page access through a novel proxy-cache aware object bundling technique. Richer content for a web page is always the result of increasing the number of embedded objects inside. However, due to the significant setup costs of a TCP connection, HTTP is known to be inefficient for transfers of small objects. As a result, request bundling techniques such as MGET and N-to-1 Bundle were proposed to eliminate the need for multiple requests by packing all associated embedded objects into a single bundle. However, these proposals all suffer from the lack of support from proxy caching. To address this problem, we firstly present the traffic characteristics of embedded object retrieval. This is then used as an argument upon which our proxy-cache aware **PC-Bundle** mechanism is built. The unique feature of our mechanism is that the advantages of all previously proposed bundle techniques can be achieved without any redundant object retrieval in the presence of proxy cache. Compared to previously proposed bundle requests such as N-to-1 Bundle, our PC-Bundle mechanism can achieve an improvement ranging from 21.2% to 34.8% for page latency reduction and 17.2% to 36.4% for page network traffic, which are very significant.

1. Introduction

Page retrieval latency is always a key metric that content delivery and distribution network service providers would like to improve. Besides the potential reuse of static web objects by proxy cache [RaS02] [Wes01], researchers are actively investigating mechanisms to accelerate the downloading process of page retrieval. This direction has huge potentials because it covers all pages and objects, independent of their cacheability.

When a user requests for a web page, its page container object, usually in the form of a HTML

file, would be returned. Then the content would be parsed chunk-by-chunk and individual requests for the embedded objects that are used by the container object will be requested. Hence, the retrieval of a web page can be broken into two stages: the retrieval of the page container object followed by its associated embedded objects. While the page container object has to be retrieved in only one manner, the retrieval of its embedded objects can be accelerated in various ways. Examples include persistent connection [FeG99] [KrR01], parallel connection [GeN98] [RoK00] [MiS99] [RoB02], and bundling [PaM94] [Fra94] [KrR01] [WiM01]. Note that while these mechanisms try to address the same latency problem, their approaches and the targets for performance optimization are quite different [KrW00]. Hence, they should be viewed as complementary (or alternative) to each other [WiM01].

In this paper, we would like to focus on the direction of bundling web objects together into a single request for more efficient content delivery. Earlier development in this direction includes the proposal for GETALL and GETLIST [PaM94] and MGET [Fra94]; the latest one is the N-to-1 bundle [WiM01]. While all these mechanisms achieve very good results, they share one important limitation. Nothing about the impact of the presence of proxy cache to their schemes is mentioned. This is a valid concern because when the bundle request arrives at the content server, all objects described in the bundle request will be sent out as one package, independent of the potential hits of some of the bundle objects in the proxy caches that are along the data transmission path. This will cause substantial performance loss, as proxy caching has been proven to be an important technology to reduce web access latency [KrR01] [RaS02].

To address the above problem, we propose a proxy-cache aware version of the bundle request, which we call **PC-Bundle**. The unique feature

of this mechanism is that the advantages of all previously proposed bundle techniques can be achieved without any redundant object transfer in the presence of proxy cache. Compared to previously proposed bundle requests such as N-to-1 Bundle, our PC-Bundle mechanism can achieve an improvement ranging from 21.2% to 34.8% for page latency reduction and 17.2% to 36.4% for page network traffic, which are very significant.

In our study, the object retrieval latency is simply the time taken required to completely transfer the object while the page retrieval latency is the aggregate of the retrieval latency of the page container object and all of its associated embedded objects. The page retrieval latency needs not necessarily be the simple sum of the respective retrieval latency of the page container object and its embedded objects. This is due to the parallelism existed in web browsers for simultaneously object fetching. Currently, the parallelism width being used in Netscape and Microsoft IE browsers is four [ChL02].

2. Existing Methods to Retrieve Embedded Objects

There are three main approaches used to accelerate the retrieval of embedded objects within a web page. They are: (i) parallel connections, (ii) persistent connection, and (iii) bundle transfer.

Some web browsers, after retrieving the page container object, establish parallel connections, one per embedded object, to retrieve the embedded objects. This is in part due to that the web browsers start parsing the initial portion of the container object that they receive and then make requests for the embedded objects, even when the retrieval of the container object is still underway. In addition, the browsers want to simultaneously render the embedded objects (especially images). This is possible with parallel connections. Thus the user would be able to view the web page in increasing quality and hence reduces the user-perceived page retrieval latency.

WebMUX [GeN98] is an experimental multiplexing protocol that allows multiple application-layer sockets to transfer data fragments using a single transport-layer TCP connection. Paraloading [RoK00] [MiS99] [RoB02] refers to the parallel segmented download (PSD) of an object using parallel connections to multiple mirrors (one connection per mirror). This is different from the convention of retrieving the object solely from one site. The user would first determine the mirrors from which the required object could be downloaded. Then, the user would select the mirrors to which a persistent connection would be established each.

The use of parallel connections, however, has some disadvantages such as additional overheads and slow transfer rates due to TCP slow-start. Also, if a container object has many embedded objects (for example more than 20), the use of parallel connections can have a negative impact on the performance and thereby increasing the page retrieval latency and not to mention user-perceived retrieval latency, which is definitely not desirable.

A persistent connection can be used, in lieu of parallel connections, for a serialized transfer of the embedded objects [FeG99] [KrR01]. Here, the server would return the embedded objects one-by-one, in the same sequence that the requests have been received. The client can also pipeline its requests so that it would reduce its waiting time for the transfers to take place. Nevertheless, the bulk of the retrieval latency comes from the transfer of the objects. The main advantage of using a persistent connection is that the client would bypass the overheads for the establishment of connections and the TCP slow-start stage that it would have incurred if parallel connections were used. It is possible that a serialized transfer of a group of embedded objects is faster than the use of parallel connections for the same purpose. However, it is reported in [CaS00] [WiM01] that support for persistent connections does not necessarily lead to reduced retrieval times for a set of objects from a server. This can be due to the

inconsistent support from web servers, user agents, and intermediaries for persistent connections, particularly with pipelining.

For bundle request, the earlier proposals are MGET, GETALL, and GETLIST [PaM94] [Fra94] [KrR01]. They are the precursor of pipelining in a persistent connection. GETALL specifies that the server returns the requested page container object as well as all the associated embedded objects (if any) to the client. As it is not cache-friendly and has performance and security issues, we would not discuss it further. MGET and GETLIST allow a user to specify a list of requests (for objects) in a single request. The server would then parse through this list and return the objects (if applicable) in sequence. While this might mean that the client would have to consolidate all the requests to construct this list (and thereby incurring some waiting time), it would allow the server to know immediately what are the objects that the client requests at one instance.

N-to-1 Bundle is the latest version of the bundle request [WiM01]. It suggests that the retrieval of objects would be shorter if they are bundled (and possibly compressed) together than that of their individual transfers. The server would advertise the availability of a bundle. This bundle would contain objects that have intra-page or inter-page relevance and in short, and most likely to be embedded objects. In addition, the bundle can be pre-determined or dynamically generated. In the case of a pre-determined bundle, the bundle might contain objects that the client does not want. On the other hand, the bundle may also lack the objects the client wants. Also, this form of bundling is not exactly cache-friendly. This is a trade-off from the associated costs and complexity of bundling on-the-fly and to the client's specifications.

3. Traffic Characteristics of Embedded Objects

In this section, we would like to examine some traffic characteristics of embedded objects based

on a set of IRCACHE proxy traces [Trace] as our dataset. This should give us an insight on how to improve upon their retrieval latency.

3.1. Number of Embedded Objects Per Page

Table 1 illustrates the breakdown of the number of objects per page. This is derived from our IRCACHE dataset from which we have extracted 2725159 pages with 1934929 embedded objects from a total of 4660088 objects.

Table 1: Some Statistics Pertaining to Number of Objects per Page (from NLANR dataset)

Number of Objects	1	≥ 2
Percentage of Entries	40.0%	60.0%
Percentage of Pages	68.4%	31.6%

While a large proportion of the web pages served is a single object by itself, about 32% have at least one embedded object. In addition, web pages with at least one embedded object amount to about 60% of the total number of entries. Hence, while there are fewer pages with at least one embedded object, they account for a larger portion of the NLANR dataset. This is consistent with the studies by [CaP05] [Mah97].

3.2. Content of Embedded Objects

Table 2 shows that images are the dominant group of embedded objects. This is followed by text resources. GIFs, JPEGs and PNGs images form the majority of the images resources while HTML objects are the most numerous among the text resources.

Table 2: Content-Type of Embedded Objects

Content-Type of Embedded Obj.	Text	Images	Audio	Video	App.	Others
% of Total	28.0%	62.4%	0.14%	0.39%	7.93%	1.15%

Given the high percentage of image resources, it is understandably from the user's perspective on the desire to have simultaneous rendering of the embedded images. The GIF, JPEG and PNG support this form of progressive display whereby the quality of the image rendered

improves as more of the image is retrieved. As such, the use of parallel connections is preferred to achieve this. A serialized transfer would not be able to achieve this type of simultaneous rendering. However, if a serialized transfer of a group of images is somehow faster than the parallel retrievals of the images, the faster retrieval might be chosen in preference over the simultaneous rendering.

3.3. Sizes of Embedded Objects

It can be observed that the embedded objects could be segmented into four main size classes namely:

- < 2KB
- 2KB - 5KB
- 5KB - 10KB
- 10KB

Table 3: Sizes of Embedded Objects

Size of Objects	< 2KB	2KB – 5KB	5KB – 10KB	> 10KB
% of Entries	56.6%	20.2%	11.24%	12.0%

Table 3 shows the various size categories of the embedded objects from our IRCACHE dataset. About 57% belongs to what we would refer to as small objects, accounting for more than half the total number of embedded objects that are in our dataset. The group of objects of sizes above 10KB represents about 12% of the total. We would refer to these objects as big objects. These big objects reflect the potential amount of the head-of-line blocking effect in transfers using a persistent connection.

3.4. Retrieval Latency of Embedded Objects

Table 4 lists the mean retrieval latency of embedded objects classified in terms of size as in Table 3. We observe that the mean retrieval latency for objects in the groups < 2KB, 2KB to 5KB and 5KB to 10KB are rather similar. In addition, objects that are below 2KB in size, have a relatively lower transfer rate than that for the other objects.

Table 4: Mean Retrieval Latency of Embedded Objects

Object Size (bytes)	< 2KB	2KB – 5KB	5KB – 10KB	> 10KB
Mean Retrieval Latency (millisecond)	736	824	980	5255

A further analysis in the mean transfer rates is listed in Tables 5 and 6. They show that objects of sizes below 2KB have relatively lower transfer rates compared to that of other groups of objects. Given that from Table 3 that objects below 2KB in size forms the majority (more than half) of the embedded objects, this suggests that it would be beneficial to reduce the mean retrieval latency of this group of objects.

Table 5: Mean Transfer Rate of Embedded Objects for Sizes Below 5KB

Object Size (bytes)	< 512	512 – 1KB	1KB – 2KB	2KB – 3KB	3KB – 4KB	4KB – 5KB
Mean Transfer Rate (KB/s)	0.38	1.23	1.39	3.06	5.45	3.96

Table 6: Mean Transfer Rate of Embedded Objects of Sizes 5KB to 10KB

Object Size (bytes)	5KB – 6KB	6KB – 7KB	7KB – 8KB	8KB – 9KB	9KB – 10KB
Mean Transfer Rate (KB/s)	5.59	8.87	6.60	6.73	9.04

The probable reasons why the small embedded objects have a relatively lower transfer rates are the following:

- overheads of connection establishment: as assumed, a fixed amount of resources is required for connection establishment. For larger objects, this cost is amortized with the relatively lengthy transfer.
- TCP slow-start: TCP slow-start confines the server to a low transfer rate to be increased gradually with more successful transmission. The transmission of small embedded objects would usually complete before the maturity of the slow-start stage.

4. Proxy-Cache Aware PC-Bundle Transfer of HTTP Messages

Bundle transfer, in abstraction, is a form of transfer coding except that it is applied to a

group of HTTP messages rather than to a message. This group of HTTP messages could be a list of requests or a list of responses. Here, we formally propose a proxy-cache aware mechanism for the bundle transfer of HTTP messages. Changes to the HTTP specifications are required.

4.1. Basic Mechanism

A client C would retrieve a web page with the container object Obj_0 from a server S along with the data transfer path with proxy caches $P_1, P_2, P_3, \dots, P_i, \dots, P_m$. From Obj_0 , C would determine the embedded objects that it would retrieve. Suppose that there are N such unique embedded objects. As such, C would have to make N number of HTTP requests to be denoted as $Req_1, Req_2, \dots, Req_n$. Instead of sending out these requests (pipelining or otherwise), C would bundle-transfer them in a manner similar to that of MGET and GETLIST. C would encapsulate these requests as a list of requests inside a HTTP request. Let this new HTTP request be denoted as REQ . Hence, REQ would contain a list of HTTP requests as its message body and a slightly different set of HTTP request and entity headers that would provide some information about the list of requests.

When REQ arrives at each proxy cache P_i (where $1 \leq i \leq m$), the encapsulated object request list in the bundle REQ will be parsed. Each requested object will be checked against the proxy cache storage. For any object Obj_i (where $1 \leq i \leq N$) that is found in the local cache, the object content will be sent back to C immediately without waiting for the other object requests in REQ . At the same time, Obj_i will be deleted from REQ before the *updated* bundle request REQ' (which is the difference between REQ and all Obj_i found in proxy cache) is forwarded to the next level of the network. Note that all objects found in the local proxy cache can be sent back to C in one single bundle reply, just like the server S .

S receives the latest REQ' and parses it to obtain

the individual requests. As S prepares the various responses, it determines whether to perform PC-bundle transfer and if so, which of the responses to be bundle-transferred. This would be a server-driven policy. In our case, S would only choose small responses. Small responses would usually refer to those responses containing small embedded objects or no entities at all. S will encapsulate these selected responses into a HTTP response to be denoted as RES . Thus RES will contain the list of selected responses as its message body and a slightly different set of HTTP entity headers that would provide some information about the list of responses. This list of responses is a subset of the responses to all the requests in REQ or REQ' . S will transfer RES first followed by the individual transfers of the other responses (to requests in REQ) which are not bundled in RES .

4.2. Assumptions

We make the following assumptions in the following three sub-sections.

4.2.1. Client-Specific Assumptions

The assumptions that are client-specific are:

- The client might issue a list of requests instead of individual requests. This is to be known as a request-list. The client suffers insignificant performance penalty in the construction of this list.
- The client will receive a bundle that is a collection of responses to some or all of its requests (in a request-list). The client will exert a fixed amount of resources to dissemble the bundle.
- There might be more than one bundle (although unlikely) received as response for each request-list. The client might receive a mixture of bundles and individual responses to a request-list.
- The client is prepared to receive responses not in the same order as that of its requests. In addition, the order of requests and responses is not important to the client.

4.2.2. Server-Specific Assumptions

The assumptions that are server-specific are:

- The server will determine whether or not to use the PC-Bundle mechanism whenever a request-list is received. The server incurs insignificant overhead for this action.
- The server will determine which of the responses to a request-list should be bundled. It will exert a fixed amount of resources to determine and assemble a bundle.
- The server should consider that only responses to requests from a request-list should be bundled.

4.2.3. Proxy-Specific Assumptions

The assumptions that are server-specific are:

- Upon receiving the request-list in a PC-bundle, the proxy has the ability to parse and understand all individual object requests.
- The proxy will remove from the request-list all objects that are found in its cache. It will form a new PC-bundle's request-list to be forwarded to the server or next proxy.
- For all the cache-hit objects in the request-list, their content will be sent to the client using the same mechanism as the server does.

4.3. Detailed Specifications of Modifications to HTTP for PC-Bundle

4.3.1. Request

The motivation for a client to PC-Bundle its requests is that the server would be able to receive all these requests at one pass. This would allow the server to PC-Bundle all applicable responses back to the client, without waiting for potential additional requests from the client for whose responses could be piggybacked onto the bundle transfer. Encapsulating multiple requests in a single message would also mean a reduction in the number of packets in the network. This would help to reduce the load on the network. At the same time, the client must be able to tolerate

some delay in the consolidation of all requests. In addition, it would take additional latency to transmit the encapsulation of requests since it would be considerably larger in size than the individual requests. (Note that the later situation is not very bad, as the encapsulated response can be interpreted in a chunk-by-chunk manner).

Here, we propose the following changes to the syntax of HTTP specifications in order to accommodate the capability of PC-Bundle of requests. The syntax of a HTTP request message looks like the following:

```
Request = Request-Line
         *(General-Header | Request-Header |
         Entity-Header)
         CRLF
         [Message-Body]
Request-Line = Method SP Request-URI
             HTTP-Version CRLF
```

4.3.1.1. Request-Line

We propose that the field Request-URI is to be made optional and omitted only when the method to use PC-Bundle is invoked. Note that in a MGET request, the Request-URI would not be required. Thus the new Request-Line would look like the following.

```
Request-Line = Method SP [Request-URI]
             HTTP-Version CRLF
```

Thus a PC-Bundle request is different from a normal HTTP request in that the Request-URI is omitted and that it has a Message-Body which is simply the concatenation of multiple HTTP requests.

4.3.1.2. Request-Header

The client would want to specify that it is able to receive a PC-Bundle type of responses. It could only do so through the use of the Accept header for the recommended new media type bundle/responses.

```
Accept: bundle/responses
```

The inclusion of this Request-Header is mandatory for the responses of the PC-Bundle to take place.

4.3.1.3. Entity-Header

The client would have to specify the following Entity-Headers:

- Content-Length: the size of the Message-Body which should be the sum of all the encapsulated requests.
Content-Type: This should be text/ASCII or a recommended new media type bundle/requests. This new media type should be only used to mean the PC-Bundle requests. The definition of the field media-type for Content-Type includes the optional input of an attribute. This option may be exercised to specify the number of requests that are encapsulated in the Message-Body of the PC-Bundle request.

4.3.1.4. Message-Body

The Message-Body of a PC-Bundle request would be simply the concatenation of the requests. These requests may be separated by a CRLF. In addition, these requests must be full-fledged requests on their own. The recipient of a PC-Bundle request should be able to parse through the Message-Body, extract the individual requests and resend them to another intermediary or origin server without (non-transparent) modifications. The Message-Header must not contain any other PC-Bundle requests.

4.3.2. Response

Only the recipient of a PC-Bundle request which includes an Accept: bundle/responses Request-Header would be in a position to use PC-Bundle responses back to the client. If the recipient does not receive a PC-Bundle request or receive a PC-Bundle request without an Accept: bundle/responses Request-Header, the

recipient must not send any PC-Bundle responses back to the client. The subsequent decision whether to use PC-Bundle or not and which of the responses to use PC-Bundle is a server-driven policy.

The motivations for using PC-Bundle as the responses include the increase in the effective retrieval latency for small responses and a reduced network load since less number of packets would be transmitted. PC-Bundle response is preferred for non-dynamic responses.

A typical HTTP Response message has the following syntax:

```
Response = Status-Line
          *(General-Header|
            Response-Header|Entity-Header)
          CRLF
          [Message-Body]
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

4.3.2.1. Status-Line

Two types of responses to a PC-Bundle request are possible. There would be either a PC-Bundle for some or all responses (to the list of requests) or none at all, meaning that the responses are returned as per normal. A "successful" PC-Bundle response would have a 2XX status code. The recommended status code is 207. The Reason-Phrase could be "Bundled Content". Hence the Status-Line of a PC-Bundle response would look like the following

```
HTTP/1.1 207 Bundled Content
```

4.3.2.2. Entity-Header

The Content-Type and Content-Length Entity-Headers are mandatory in a PC-Bundle response. The value to be assigned to Content-Type would be the media-type bundle/responses. As with the Content-Type header in the PC-Bundle

Request, there is an optional input of an attribute for the media-type field. This option may be exercised to specify the number of responses that are encapsulated in the Message-Body.

4.3.2.3. Message-Body

The Message-Body is the concatenation of various responses that are selected for PC-Bundle. These responses may be separated using a CRLF. An additional Entity-Header URI must be included in every response that has a Message-Body. It will be used to identify the entity in the Message-Body. The value of URI must correspond to that of the Request-URI listed in the Request-Line of the request. The Message-Body must not contain any other PC-Bundle responses.

5. Experimental Results on PC-Bundle Mechanism

In this section, we would like to highlight the potentials of our PC-Bundle mechanism through trace-driven stimulation. In particular, we would like to compare our mechanism with the latest N-to-1 Bundle proposal [WiM01] in the presence of proxy cache. To make the study more realistic, the performance metric is chosen to be the total page latency instead of the object retrieval latency to reflect the effect of parallel object fetching to the actual performance. We also try to make our simulation environment and assumptions as close as possible to those in the latest N-to-1 Bundle proposal.

5.1. Environment of Stimulation

Our study is done in a trace-driven stimulation using three sets of proxy traces from NLANR [IRCA], Digital [Mog96] and Berkeley [Berk]. The objects from the entries in these three proxy traces are grouped into pages and the traffic characteristics of these pages are examined. *Total page latency* refers to the total time required to retrieve the pages in each set of proxy traces while *total page traffic* refers to the

effective bandwidth consumed in the retrieval of these pages. In our sensitivity study, we shall look at three cache parameters. They are: (i) cache size (as a percentage of total size of unique objects in the trace), (ii) replacement policy, and (iii) degree of parallelism for simultaneous object fetching.

To make a fair comparison, our stimulation emulates the environment set in [WiM01]. Bundling makes use of the best transfer rate among that of the objects in a page to be the effective transfer rate of the page. In the event of a partial page hit (the container page or some embedded objects are misses), the page is treated as a complete page miss. The effective transfer rate of the page used is then the best transfer rate among that of the object misses. Likewise, in the event of a complete page (the container page and the embedded objects are all hits), the best transfer rate among that of these hits is chosen as the effective transfer rate of the page. The page latency is then the time taken to transfer the amount of data of size equal to the sum of the sizes of the container page and its embedded object, using the effective transfer rate. In our PC-Bundle mechanism, it has two transfer rates. It uses the best transfer rate among that of the object hits and the best transfer rate among that of the object misses. The page latency is then the time taken to retrieve the bundle of hits and the bundle of misses respectively, using their respective effective transfer rates.

Page traffic effectively refers to the amount of data transferred for the retrieval of a page. For bundling in the case of a partial page hit or a complete page miss, the page traffic of this page is effectively the sum of the size of the page container object and all its associated embedded objects. In the case of the normal proxy settings and our PC-Bundle mechanism, the page traffic could be smaller in the case of page/object hits (where only validation and not the entire object/page is needed). Thus, for our study, we can regard the page traffic for the normal proxy settings and that of our PC-Bundle mechanism to be the same.

Table 7: Effect of Cache Size on Total Page Latency (GDS, Parallelism = 4)

Cache size	N-to-1 Bundle/Normal			PC-Bundle / N-to-1 Bundle			PC-Bundle/Normal		
	NLANR	Digital	Berkeley	NLANR	Digital	Berkeley	NLANR	Digital	Berkeley
5%	56.0%	24.9%	38.4%	21.2%	22.7%	27.4%	65.3%	41.9%	55.3%
10%	55.3%	22.4%	35.5%	22.3%	25.0%	31.3%	65.3%	41.9%	55.7%
15%	55.3%	21.5%	34.1%	22.9%	26.2%	33.6%	65.6%	42.0%	56.2%
20%	55.0%	20.9%	33.3%	23.5%	27.0%	34.8%	65.6%	42.2%	56.5%

In our study, the standard cache configuration and network environment for reference is:

- Cache size equal to 10% of the total size of unique objects in the trace under study
- Replacement policy being set to Greedy-Dual-Size [ArW97][ArF98][Cal97] [LoR98]
- Parallelism width for simultaneous object fetching being set to 4 (typical in current browsers including Netscape and IE [ChL02])

Each of these parameters will also be varied so that the sensitivity of our results and conclusion on each of these parameters can be investigated.

5.2. Results With Respect to Cache Size

The size of a proxy cache effectively determines the number and size of objects that could be stored in the cache. In our study, we express the cache size as a percentage of the total size of unique objects in the dataset under examination. The values used in this study are 5%, 10%, 15% and 20%. To examine the effect of cache size on Total Page Latency and Total Page Traffic, we set the cache replacement policy to be the Greedy Dual-Size (GDS) algorithm and the degree of parallelism to 4.

5.2.1. Cache Size on Total Page Latency

Table 7 shows that the N-to-1 Bundle and PC-Bundle improves significantly on the Total Page Latency and that PC-Bundle outperforms N-to-1 Bundle. It shows that N-to-1 Bundle improves the Total Page Latency by about 55% for the NLANR dataset while for the Digital and Berkeley datasets, the it is about 20% to 38%. This difference could be explained by that the NLANR proxy is a public proxy while the Digital and Berkeley each served a specific user group. As such, the NLANR dataset would contain relatively a

lesser number of partial page hits and hence the negative effects of N-to-1 Bundle on caching are less influential on the NLANR dataset. This is an indication of the influence that N-to-1 Bundle has over public proxies. We also observe that as the cache size is increased, N-to-1 Bundle degrades in performance. This should be due to that as the cache size is increased, there would be more partial page hits.

Compared to N-to-1 Bundle, PC-Bundle has a much superior performance. It outperforms the N-to-1 Bundle by about 25% for Digital, 30% for Berkeley, and 22% for NLANR. And these results are quite insensitive to the cache size. This is not surprising, as the performance gain comes from the partial hit pages.

5.2.2. Cache Size on Total Page Traffic

Table 8 shows that while N-to-1 Bundle gives a better Total Page Latency than the normal case does, it consumes excessive memory bandwidth. The increase in Total Page Traffic can range from about 18% to 36%, which is very significant. On the other hand, although PC-Bundle gives even better Total Page Latency performance than N-to-1 Bundle, it does not cause any additional bandwidth consumption even when compared to the normal case. This shows its superiority and practicability over N-to-1 Bundle.

Table 8: Effect of Cache Size on Total Page Traffic (GDS, parallelism = 4)

Cache Size	N-to-1 Bundle/(Normal or PC-Bundle)		
	NLANR	Digital	Berkeley
5%	17.2%	27.3%	24.6%
10%	18.4%	33.2%	30.2%
15%	19.3%	36.8%	33.8%
20%	20.5%	39.4%	36.4%

Table 9: Effect of Cache Replacement Policy on Total Page Latency (Cache Size = 10%, Parallelism = 4)

Cache Replacement Policy	N-to-1 Bundle / Normal			PC-Bundle / N-to-1 Bundle		
	NLANR	Digital	Berkeley	NLANR	Digital	Berkeley
GDS	55.3%	22.4%	35.5%	22.3%	25.1%	31.3%
LFU	54.2%	28.6%	44.1%	26.2%	22.4%	27.5%
LRU	55.0%	31.6%	18.0%	25.2%	20.9%	25.3%

As the cache size is increased, the Total Page Traffic from N-to-1 Bundle remains relatively constant while that of the normal proxy settings and PC-Bundle decreases. This decrease is most probably accounted for by the increase in the number of partial page hits in the corresponding increase in the cache size. This indicates an effective increase of Total Page Traffic for N-to-1 Bundle as the cache size is increased.

The additional page traffic from the Digital and Berkeley datasets using N-to-1 Bundle amount to about 24% to 40% respectively while that for the NLANR dataset is less than 20%. This reinforces our observation that there are relatively more partial page hits in the Digital and Berkeley datasets than in the NLANR dataset.

5.3. Results With Respect to Cache Replacement Policy

The cache replacement policies that are used in this study are the Greedy Dual-Size (GDS), Least-Frequently-Used (LFU) and Least-Recently-Used (LRU) algorithms. The cache parameters, cache size and degree of parallelism, are set to 10% and 4 respectively.

5.3.1 Cache Replacement Policy on Total Page Latency

Table 9 shows that the superior performance of PC-Bundle over N-to-1 Bundle is quite insensitive to the cache replacement policy. Similar performance gain of 22% to 27% is still observed for PC-Bundle. Further, it shows that LFU in the NLANR dataset achieves the lowest Total Page Latency while it is so for GDS in both the Digital and Berkeley datasets.

Correspondingly, with reference to Table 9, N-to-1 Bundle performs the worst with LFU in the NLANR dataset and with GDS in both Digital and Berkeley datasets while PC-Bundle has the best performance over N-to-1 Bundle in the exact same order. Thus, the relative performance of PC-Bundle over N-to-1 Bundle, using different cache replacement policies, corresponds inversely to the performance of N-to-1 Bundle.

5.3.2. Cache Replacement Policy on Total Page Traffic

As illustrated in Table 10, the Total Page Traffic for N-to-1 Bundle remains relatively constant regardless of the cache replacement policy used. GDS in the NLANR dataset accounts for a larger Total Page Latency for that of normal proxy settings and PC-Bundle while it is the case for LRU in both the Digital and Berkeley datasets although the differences here are very slight. In the case of the NLANR dataset, PC-Bundle is most effective when LRU is used and least effective when GDS is used. In the other two datasets, the choice of the cache replacement policy does not seem to have a significant effect on the effectiveness of PC-Bundle over N-to-1 Bundle. All these result are expected, just like the situation of changing cache size.

Table 10: Effect of Cache Replacement Policy on Total Page Traffic (Cache Size = 10%, Parallelism = 4)

Cache Replacement Policy	N-to-1 Bundle / (Normal or PC-Bundle)		
	NLANR	Digital	Berkeley
GDS	18.4%	33.2%	30.2%
LFU	27.0%	33.7%	29.6%
LRU	26.5%	31.8%	28.0%

Table 11: Effect of Parallelism on Total Page Latency (Cache Size = 10%, GDS)

Degree of Parallelism	N-to-1 Bundle/Normal			PC-Bundle/ N-to-1 Bundle		
	NLANR	Digital	Berkeley	NLANR	Digital	Berkeley
4	55.3%	22.4%	35.5%	22.3%	25.1%	31.3%
8	55.0%	21.7%	34.9%	22.0%	24.5%	30.2%
16	54.9%	21.5%	34.9%	21.9%	24.4%	29.9%
32	54.9%	21.5%	34.8%	21.9%	24.4%	29.9%

5.4. Results With Respect to Degree of Parallelism

We assume that there is a degree of parallelism in all web page retrievals. The degrees of parallelism examined in this study are 4, 8, 16 and 32. The cache parameters, cache size and cache replacement policy, are set to 10% and GDS respectively. As the degree of parallelism does not have any effect on the Total Page Traffic, we would not be discussing this aspect.

5.4.1. Parallelism on Total Page Latency

Table 12: Effect of Various Degrees of Parallelism on Total Page Latency (NLANR Dataset, Cache Size = 10%, GDS)

Degree of Parallelism (DOP)	Relative Improvement in Total Page Latency over Previous DOP
1	N/A
2	16.5%
3	3.69%
4	1.36%
5	0.71%
6	0.38%
7	0.22%
8	0.18%
9	0.11%
10	0.10%

Table 11 shows that as the degree of parallelism is increased, the effectiveness of N-to-1 Bundle and PC-Bundle over N-to-1 Bundle decreases very slightly. This indicates that an increase in the degree of parallelism does not significantly affect the effectiveness of N-to-1 Bundle and PC-Bundle. A probable explanation for this is that the basic degree of parallelism (4) has already significantly reduced Total Page Latency so much so that further increases in the degree of parallelism would only return marginal improvements. This is verified by Table 12 which shows the relative improvements in Total Page Latency using

various degrees of parallelism on the NLANR dataset.

6. Conclusions

In this paper, we propose a proxy-cache aware bundling mechanism, the PC-Bundle, to accelerate the downloading process of embedded objects in a web page. PC-Bundle is a form of HTTP transfer encoding applied to a group of objects (in this case embedded objects) to effectively reduce the total number of TCP data packets and hence reduce their effective retrieval latency. The formal specifications of the PC-Bundle mechanism are presented. To show its effectiveness, we perform trace simulation study to compare our PC-Bundle with the latest bundling solution, the N-to-1 Bundle under varying cache environments using three sets of public proxy traces. Our result shows that:

- Using the same assumptions and environment that of N-to-1 Bundle, PC-Bundle outperforms N-to-1 Bundle in terms of Total Page Latency and Total Page Traffic by about 20% to 25%.
- An increase in the cache size will result in a decrease and an improvement in the effectiveness of N-to-1 Bundle and PC-Bundle in terms of Total Page Latency respectively. In addition, N-to-1 Bundle will have a corresponding increase in Total Page Traffic.
- LFU seems to favor public proxies (such as NLANR) using PC-Bundle in terms of both Total Page Latency and Total Page Traffic. In addition, applying GDS to public proxies would reduce slightly the effectiveness of PC-Bundle over N-to-1 Bundle.
- In proxies which are more centric to a group of users (Digital and Berkeley), the choice

of the cache replacement policy does not significantly affect the effectiveness of N-to-1 Bundle and PC-Bundle.

- The degree of parallelism has only a slight influence on the Total Page Latency.

References

- [ArF98] Arlitt, M., Friedrich, R., Jin, T., "Performance Evaluation of Web Proxy Cache Replacement Policies," *Technical Report HPL-98-97, HP Laboratories*, June 1998.
- [ArW97] Arlitt, M., Williamson, C., "Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers," *Simulation Journal*, Vol. 68, No. 1, January 1997, pp. 23-33.
- [Berk] UC Berkeley Web Proxy Traces, <http://ita.ee.lbl.gov/>
- [CaI97] Cao, P., Irani, S., "Cost-Aware WWW Proxy Caching Algorithms," *Proceedings of USENIX Symposium on Internet Technologies and Systems*, December 1997.
- [CaP95] Catledge, L.D., Pitkow, J.E., "Characterizing Browser Strategies in the World Wide Web," *Computer Networks and ISDN Systems*, 26(6), 1995, pp.1065-1073.
- [CaS00] Cardwell, N., Savage, S., Anderson, T., "Model TCP Latency," *Proceedings of the 2000 IEEE INFOCOM Conference*, Tel-Aviv, Israel, March 2000.
- [ChL02] Chi, C.H., Li, X., Lam, K.Y., "Understanding the Object Retrieval Dependence of Web Page Access," *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2002.
- [FeG99] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1," *IETF RFC2616*, June 1999.
- [Fra94] Franks, J., "Proposal for an HTTP MGET Method," 1994.
<http://ftp.ics.uci.edu/pub/ietf/http/hypermil/1994q4/0260.html>
- [GeN98] Gettys, J., Nielsen, H.F., "The WebMUX protocol," *Expired Internet Draft*, August 1998.
<http://www.w3.org/Protocols/MUX/WD-mux-980722.html>
- [IRCA] IRCACHE Proxy Traces, <http://ircache.nlanr.net>.
- [KrR01] Krishnamurthy, B., Rexford, J., *Web Protocols and Practice*, ISBN 0-201-71088-9, Addison-Wesley, 2001.
- [KrW00] Krishnamurthy, B., Willis, C.E., "Analyzing Factors that Influence End-to-End Web Performance," *Proceedings of the 9th World Wide Web Conference*, April 2000.
- [LoR98] Lorenzetti, P., Rizzo, L., "Replacement Policies for a Proxy Cache," *IEEE/ACM Transactions on Networking*, Volume 8, Issue 2, April 2000.
- [Mah97] Mah, B., "An Empirical Model of HTTP Network Traffic," *Proceedings of IEEE INFOCOM Conference*, April 1997.
- [MiS99] Miu, A., Shih, E., "Performance Analysis of a Dynamic Parallel Downloading Scheme from Mirror Sites Throughout the Internet," Term Paper, *LCS MIT*, December 1999.
- [Mog96] Mogul, J., *Digital's Web Proxy Traces*, 1996.
- [PaM94] Padmanabhan, V.N., Mogul, J.C., "Improving HTTP Latency," *Proceedings of the 2nd International World Wide Web Conference*, Oct. 1994. Also in *Computer Networks and ISDN Systems*, 28(1/2), December 1995, pp. 25-35.
- [RaS02] Rabinovich, M., Spatscheck, O., *Web Caching and Replication*, ISBN 0-201-61570-3, Addison-Wesley, 2002.
- [RoB02] Rodriguez, P., Biersack, E.W., "Dynamic Parallel-Access to Replicated Content in the Internet," *IEEE/ACM Transactions on Networking*, August 2002.
- [RoK00] Rodriguez, P., Kirpal, A., Biersack, E.W., "Parallel-Access for Mirror Sites in the Internet," *Proceedings of IEEE INFOCOM 2000 Conference*, March 2000.
- [Trace] IRCACHE proxy traces from <http://ircache.nlanr.net:pb.sanitized-access.20020818-24>
- [Wes01] Wessels, D., *Web Caching*, ISBN 1-56592-536-X, O' Reilly & Associates, 2001.
- [WiM01] Wills, C.E., Mikhailov, M., Shang, H., "N for the Price of 1: Bundling Web Objects for More Efficient Content Delivery," *Proceedings of the 10th International World Wide Web Conference*, May 2001.