Exploiting Result Equivalence in Caching Dynamic Web Content

Ben Smith, Anurag Acharya, Tao Yang, and Huican Zhu
Department of Computer Science, University of California
Santa Barbara, CA 93106

Abstract

Caching is currently the primary mechanism for reducing the latency as well as bandwidth requirements for delivering Web content. Numerous techniques and tools have been proposed, evaluated and successfully used for caching static content. Recent studies show that requests for dynamic web content also contain substantial locality for identical requests. In this paper, we classify locality in dynamic web content into three kinds: *identical* requests, *equivalent* requests, and *partially equivalent* requests. Equivalent requests are *not* identical to previous requests but result in generation of identical dynamic content. The documents generated for partially equivalent requests are *not* identical but can be used as temporary place holders for each other while the real document is being generated. We present a new protocol, which we refer to as *Dynamic Content Caching Protocol (DCCP)*, to allow individual content generating applications to exploit query semantics and specify how their results should be cached and/or delivered. We illustrate the usefulness of DCCP for several applications and evaluate its effectiveness using traces from the Alexandria Digital Library and NASA Kennedy Center as case studies.

1 Introduction

As many Web sites evolve to provide sophisticated e-commerce and personalized services, dynamic content generation becomes more popular. Using multi-processor or cluster-based servers can speedup resource-intensive dynamic content generation [6, 12, 24]. If successful, caching can provide significant additional benefit by reducing server load, end-to-end latency and bandwidth requirement. Numerous techniques and tools have been proposed, evaluated and deployed for caching static content. There has been recent interest in caching dynamic Web content as well [5, 7, 10, 14, 16]. Dynamic content has three forms of locality: *identical* requests, *equivalent* requests, and *partially equivalent* requests.

- **Identical requests:** These requests have identical URLs which result in the generation of the same content. Recent studies [14, 16] have shown that this locality can be successfully exploited for caching.
- **Equivalent requests:** The URLs of these requests are syntactically *different* but result in generation of *identical content*. For example, map servers (e.g. [13, 23]) frequently impose a grid on the coordinate space and return the same map when presented with any location within a given region in the grid. As another example, news servers may wish to return different front-pages for requests from users from different regions or different sites. In this case, requests from all clients in a group are equivalent.
- **Partially equivalent requests:** These requests are syntactically *different* but result in generation of content which can be used as a *temporary place-holder for each other*. For example, documents which are conditionally distilled to a lower-resolution version by a service (e.g., TranSend [12]) can be used as a place-holder for the originals. As another example, maps(e.g., MapQuest [13]) and aerial images (e.g., the TerraServer [17])

with more than a given degree of overlap could be used as placeholders for each other.

Exploiting similarity between dynamic documents has been shown to be beneficial for fast Web page delivery based on delta-encoding [4, 15, 18]. In this paper, we present a new protocol, which we refer to as the *Dynamic Content Caching Protocol (DCCP)*, to allow individual content generating applications (e.g. CGI applications, Java servlets etc) to specify equivalence between different GET-based requests they serve. This information can be used by web caches to exploit these additional forms of locality. Identical requests and equivalent requests can directly fulfilled using previously cached results. For partially equivalent requests, previously cached content can be immediately delivered as an approximate solution to the client while actual content is generated and delivered. This allows clients to browse partial or related or similar results promptly while waiting for more accurate information.

We have designed DCCP using the extension mechanism provided in HTTP 1.1 cache control directives [11]. This has several advantages. First, specifying result equivalence information as an extension of the HTTP header allows DCCP to be deployed incrementally — servers, proxies and individual content generating applications can be upgraded individually. In many cases, existing CGI scripts can be upgraded to generate these headers simply by using a short Perl or shell script as a wrapper. Second, since HTTP 1.1 specifies that headers *must* be propagated by compliant caches, DCCP directives can be expected to reach most caches. Finally, a declarative header-based specification of caching requirements allows value-added proxies such as *TranSend* [12] to compose server-provided directives and hints with their own. For example, the TranSend proxy provides a distillation service for heterogeneous clients. A distilled version of a web document is a lower-resolution (or a summary) version of the original and can be returned in response to a request for the original document. Since this functionality is provided by an intermediate node, the ability to augment and compose cache control specifications allows the results of such processing to be effectively cached by web caches closer to the client.

This paper is organized as follows. We first summarize the related work and then present a description of DCCP. We illustrate its utility for a variety of applications including a customizable news service, a location-dependent weather information service, server-side image maps, and a geographically indexed digital library. Finally, we discuss our current cache design and implementation for DCCP and evaluate the effectiveness of DCCP using real traces from the Alexandria Digital Library and NASA Kennedy Center and a synthetic trace for a weather forecast application.

2 Related Work

Exploiting similarities between dynamic documents generated by multiple invocations of the same web application was first proposed in [4, 15] for delta-encoding. This idea was explored further by Mogul et al. [18] as *query clustering*. They analyzed web proxy and packet-level traces to evaluate the extent of similarity between dynamic documents with the same "URL prefix" (usually identifying the generating application) but different suffixes (usually arguments to the generating application). They found that there is substantial locality in "URL prefixes" – the 100780 distinct dynamic content requests in their proxy trace were based on just 12004 prefixes. They also found that exploiting such similarities in delta-encoding can offer significant performance advantages. The DRP protocol [21] uses a checksum algorithm (e.g. MD5) to identify file equivalence and avoid unnecessary file download if checksum values do not change. These proposals place the responsibility of creating equivalence (or partial equivalence) classes on a caching agent and a server, with little or no support from the applications. DCCP, on the other hand, allows individual web applications to explicitly specify equivalence between different documents they generate, which can help delta-encoding in identifying similar documents.

The idea of partial result delivery has been considered by Dingle et al. [9]. They propose that a caching agent could send previously cached data to a client so she/he could browse an old version while current data is being fetched. Optimistically transferring potentially out-of-date data to reduce end-to-end latency is also considered by Banga et al. [4] in delta-encoding. In this work, they propose sending the cached version of a dynamic document to the requesting agent (client or the next level proxy) while a delta is being fetched. DCCP lets application users *explicitly* define partial equivalence between the cached results and a new query and we can utilize the infrastructure of [9] or [4] to achieve partial result delivery in the implementation of a DCCP-aware caching agent.

Cao et al. [5] propose that a piece of Java code (a *cache-applet*) be attached to each dynamic document. This code is run whenever a request is received for a cached document. Cache applets can rewrite the cached document, return the cached document or direct the cache to refetch or regenerate the document. This approach, referred to as *Active Cache*, is extremely flexible and can be used to maintain consistency in an application-specific manner as well as dynamically modify existing documents. However, the flexibility of Active Cache comes with a price. It requires starting up a new Java process (virtual machine or compiled code) for every request. Keeping track of result equivalence requires creation and maintenance of a pattern-matching network. Since the cache-applet for each document is independent (for security reasons), cache-applets used to implement result-equivalence-based caching would need to save and restore its pattern-matching network to persistent storage. Depending on the access pattern, this can be a significant performance limitation. In contrast with the generality of Active Cache, DCCP is more restrictive. The trade-off is that this design simplifies implementation and provides opportunities for optimization. The limited scope and the declarative nature of the DCCP directives alleviates security concerns. Using a single global pattern-matching network allows processing for related patterns to be shared. Finally, since the structure and the function of the pattern-matching network is known to the cache, it can maintain portions of the network in memory.

Iyengar et al. [16, 7] propose that cache servers export an API that allows individual applications to explicitly cache and invalidate application-specific content. These techniques are developed for caching identical requests at original content providers and may not be feasible for proxy or client caches. These techniques can be integrated with DCCP when DCCP is deployed at server-sites.

In previous research, we have considered cooperative caching for dynamic web content on a cluster of servers [14]. The research focus is to study how clustered nodes collaborate with each other in terms of caching and the Time-To-Live method is used for maintaining result consistency. However, this work uses complete URLs as names for documents and has no notion of equivalent or partially equivalent requests.

Douglis et al. [10] propose that servers should make the structure of a dynamic document available to caches and that caches should construct the desired document on demand. The idea is that a dynamic document be specified as a static part, one or more dynamic parts and a macro-based template that combines them. The static parts and the template can be cached whereas the dynamic parts are fetched anew for every request.

3 The Dynamic Content Caching Protocol (DCCP)

The goal of DCCP is to let web applications exploit query semantics and specify equivalence or partial-equivalence between the dynamic documents they generate. DCCP has been defined using the extension mechanism provided in HTTP 1.1 cache control directives [11]. Figure 1 presents the proposed directives (see Section 4 for examples). Each document can contain one or more equivalence directives. Each equivalence directive specifies the set of requests that this document can be used to fulfill. The set of requests is specified using a pattern over the set of arguments embedded in the URL, client cookies, and the domain name and IP address

directive	:=	*eq_result* | *sim_result*
eq_result	:=	`equivalent_result =`*condition*
sim_result	:=	`partial_result=` *condition*
condition	:=	*arg* = *pattern* | (*arg* = *pattern*) *op condition*
arg	:=	`__domain` | `__IP_address` | *field_name* | *cookie_name*
cookie_name	:=	`cookie:`*field_name*
op	:=	"`&&`" | "`||`"
pattern	:=	*range* | *regexp*
range	:=	[*number*, *number*]
regexp	:=	*"Perl syntax"*

Figure 1: DCCP syntax for specifying equivalence directives. Field names are those argument names in a GET-based query.

of the machine from which the query is originated. There is no authentication requirement implied in these arguments – caches implementing DCCP are expected to provide best-effort values for these arguments.

The language used to specify the equivalence patterns provides equality for all arguments and range operators for numeric arguments. For example, "[0.0,100.0]" specifies any real number between and including 0 and 100. Patterns can be composed using logical operators ("&&" stands for "`and`" and "||" stands for "`or`"). The current version of DCCP also includes regular expressions over alphanumeric and special characters for string arguments as an experimental feature. We use Perl syntax [22] for regular expressions. Note that a directive need not specify values for all arguments. When a cache agent (e.g. a proxy server) receives a request, it extracts the application arguments from the URL, and the identity arguments from request header as well as the identity of the requesting machine. Note that DCCP has been designed for handling GET-based queries.

Since dynamic documents are generated on demand, they usually have greater consistency requirements than static documents. HTTP 1.1 provides several cache control directives to manage consistency. The current version of DCCP does not propose additional directives for this purpose. We believe more experience with dynamic documents is needed to determine the set of commonly used consistency requirements. If necessary, we could add directives to specify options such as polling periodically. The main problem with detailed consistency directives is that a proxy may not implement the directives which can result in incorrect results being delivered to clients. Thus our current focus is to support the class of applications which can benefit from DCCP with the existing HTTP 1.1 consistency mechanisms.

4 Applications

In this section, we illustrate how DCCP can be used to specify result equivalence for a variety of content generating applications.

Alexandria Digital Library: The Alexandria Digital Library (ADL) [3, 20] provides geo-spatially-referenced access to large classes of distributed library holdings. Current ADL collections contain more than 7 million entries of maps, satellite images, aerial photographs, text documents, and scientific data sets. Each entry has geographical extent represented as a minimum bounding rectangle in longitude and latitude and is spatially indexed. Queries used in the current version of ADL look like this: `GET /cgi-bin/draw_map?lat=36.81818181&lon=-115.45454545&ht=75.0&wd=180.0 HTTP/1.1`.

Consider the scenario where the ADL server knows that the maps it is serving have a limited resolution. In that case, it can use the DCCP equivalence directives to define regions contained in the same map as equivalent. For example, for the above query, it could reply:

```
HTTP/1.1 200 OK
```

```
Server:  Swala/1.8a
Content-type:  text/html
Cache-control:
equivalent_result=''lat=[36,37]&&lon=
[-115,-116]&&ht=[74,76]&&wd=[179,181]''
```

The caching agent would associate this pattern with the map. If a subsequent request is received with arguments that match this pattern, the caching agent can return that result instead of calling the application and re-executing the request. We evaluate the impact of using DCCP for ADL dynamic content requests in Section 6.1.

Server-side image maps: Image maps [19] are used to provide intuitive and attractive labels for web links. Server-side image maps consist of images that is embedded in a web page. When a user clicks on a portion of the image, the browser submits a `GET` request to the server and which includes the screen coordinates as parameters. Client-side image maps can also be used to translate coordinates within simple image regions into appropriate HTML links. A client-site browser does such a translation and thus client-site image maps are preferable in terms of performance. However, server-side image maps are useful in cases where the image map is too complicated for a client-side image map or a server needs to process clicked screen coordinates. There are a number of high volume sites including `latimes.com`, and `washingtonpost.com` use the server-site image maps.

The typical number of regions on an image map is fairly small (e.g., from five to fifteen) while the possible number of coordinates is usually at least two orders of magnitude greater. Specifying result equivalence for all coordinates using DCCP can significantly improve the performance of server-side image maps. A cache without equivalence matching would be required to store all combinations of coordinates to generate a significant number of cache hits. With DCCP, the equivalent result already present in a cache can be identified without involving the server. We evaluate the impact of using DCCP for image maps in Section 6.2.

Weather Service: Weather servers use dynamically generated content to provide up-to-date weather information for the requested location. Typically location information, zip code or city name, is encoded in the request URL. Such requests, however, can be overly precise. Twenty zip codes might have the same weather information, but since their URLs are different, a cache with no support for result equivalence would require twenty cache entries to provide complete coverage. With DCCP, a weather server can specify that the same page should be returned for all requests in the same region. Figure 2 provides an illustration for zipcodes in and around the University of California, Santa Barbara. The request is `GET /cgi-bin/weather.cgi?zip=93101 HTTP/1.1`.

```
HTTP/1.1 200 OK
Server:  Swala/1.8a
Content-type:  text/html
Cache-control:  equivalent_result=
''zip=93111||zip=93106||zip=93117''
```

Figure 2: DCCP directives for a weather application

Customized news services: There are several forms of customized news services. The simplest of these customize the content based on the location of the requesting clients. This is similar to the different editions that newspapers publish targeting specific regions. For such a scenario, all requests for a news page from the same geographical region (or Internet domain) would be equivalent. For example, all clients from `.uk` machines would be presented with United Kingdom specific news stories. The first time the page is accessed, the server must run the application to generate the customized web page; subsequent requests for the same page from machines with a `.uk` domain name can be serviced from the cache. The DCCP directive for this case is presented in Figure 3.

News services with greater degree of cus-

```
HTTP/1.1 200 OK
Server:  Swala/1.8a
Content-type:  text/html
Cache-control:  equivalent_result=
''__domain=*.uk''
```

Figure 3: DCCP headers for a UK-specific web page

```
HTTP/1.1 200 OK
Server:  Swala/1.8a
Content-type:  text/html
Cache-control:  equivalent_result=``sports=yes&&clinton=yes&&movies=yes''
Cache-control:  partial_result=``sports=yes&&clinton=yes''
Cache-control:  partial_result=``sports=yes&&movies=yes''
Cache-control:  partial_result=``clinton=yes&&movies=yes''
```

Figure 4: DCCP directives for a user-profile-specific web page

tomization could use previously stored profiles to customize the news contents for individual users. Such a service can facilitate caching of its documents by marking documents generated for users with identical profiles as *equivalent* and documents generated for users with substantially overlapping profiles as *partially-equivalent*. Figure 4 provides an illustration.

5 Design of a DCCP-aware cache prototype

The primary challenge for the design and implementation of a DCCP-aware caching agent is to be able to efficiently maintain and use information about equivalence and partial equivalence of cached documents. We have developed a multi-stage pattern-matching network for this purpose based on the Swala cooperative web cache [14]. In this section, we describe the structure of this network and how the insertion and search operations are implemented.

In the DCCP model, each application specifies its result equivalence independently. As a result, a dynamic document generated by an application can be specified to be equivalent only to other documents generated by the same application. Accordingly, our prototype maintains a separate pattern-matching network for every application. Each application is identified by its *net-path-name*. The net-path-name for a URL consists of the longest prefix that contains no arguments. Our prototype uses a hash table (with net-path-names as keys) to quickly retrieve the pattern-matching network corresponding to a URL.

We discuss how we can build an efficient pattern matching network for each application with the same net-path-name as follows. In general a pattern matching rule can be transformed into a sequence of conjunctive pattern phrases connected by logical OR. For example, pattern `"map=USA&&lat=[34,37]&&lon=[-119,-117]` ||`map=USA&&city=foo"` contains two conjunctive phrases. Given a set of conjunctive phrases derived from the same rule or from different rules, we partition them into a set of clusters and each cluster has conjunctive phrases with the same argument names used in their exact match and range match patterns (discussed below). For example, two pattern phrases ``` ``map=USA&&zip=93111'' ``` and ``` ``map=USA&&zip=93016'' ``` are in the same pattern phrase cluster.

We discuss how each pattern cluster arranges its data structure for result matching. Given conjunctive pattern phrases within each cluster, we classify patterns into three parts in increasing degrees of complexity and searching within each cluster contains three stages:

- **Exact match**. All patterns are of form `argument=val`. These patterns can be hashed together. Searching within this cluster can start from the corresponding hash table so that only results satisfy those exact match patterns are candidates for further searching.
- **Range match**. All patterns are of form `argument=[val1, val2]`. Searching for a result under these conditions is similar to the problem of point intersection searching [2]: given a set of objects specified with multi-dimensional intervals, find an object that contains a query point. In our current implementation, we have used the R*tree data structure to group all such range match patterns together.

- **Complicated string match**. All patterns are of form `argument= regular_expression_pattern`. These patterns contain complicated string matching and so far we have not found a good way to organize a set of such patterns for efficient searching. Our current solution is to linearly scan such rules to find one result that matches these string matching patterns.

Thus the first stage of searching within each cluster is to use a hash table to narrow the searching space, i.e. find potential results which satisfy all exact match conditions. The second stage of searching is to use a point intersection data structure to further narrow the searching scope. The last stage of searching linearly scans all candidate results after the above two-stage filtering.

Each pattern network for an application with the same net-path-name consists of a layered graph with a single designated root node. Each child of this root node is a subgraph corresponding to each pattern cluster. Each cluster subgraph contains a hash table for exact match patterns and data structures for range match patterns. The cached dynamic documents constitute the leaves of this graph. A dynamic document is attached to a node in the graph if and only if the sequence of tests occurring on the path from the root to this node successfully satisfies all conditions specified by the DCCP rule of this document. Note that since each document can have multiple DCCP directives and since each directive can have multiple conjunctive-phrases, a cached document can be associated with multiple interior nodes in the pattern-matching network.

When a new DCCP-annotated dynamic document is inserted into the cache, the caching agent extracts the DCCP directives and the URL of the request that generated the document. It parses the URL to extract the *net-path-name* for the request and uses it to find the root of the associated pattern-matching network. It parses the DCCP directives and transforms them into a set of conjunctive-phrases and inserts each conjunctive-phrase into the network. Since cached dynamic documents may become stale and the rule for the same URL may change, the pattern network needs to be updated periodically.

When a new URL query arrives from a client, the prototype parses the URL to extract *net-path-name* for the request and uses it to find the root of the associated pattern-matching network. It then extracts the arguments from the URL and the cookies from the header. It also extracts the IP address of the requesting machine by querying the socket structure and does a reverse lookup to determine the domain name of the requesting machine. It then uses these values to perform tests against the pattern-matching network. Notice only some of query parameters required by the network are extracted for the matching purpose. If the match is successful, that is, the match operation reaches the node corresponding to a valid cached document, the corresponding document is deemed equivalent to the one requested by the client and is returned in response to the query. Notice that there may be multiple cache entries matching a new query and the system returns the first matchable cache entry it finds.

Our above optimization is targeted at rules that mainly use exact match or range patterns. It is possible that for complex patterns (particularly those making liberal use of regular expressions), trying to determine result equivalence could take an inordinately long time. To handle this case, we suggest that the system should limit the amount of time spent searching for a single request. Since result equivalence is only a hint, terminating the search early and fetching a new copy of the document from the server is safe.

For implementing progressive delivery of partially-equivalent results, we can use the `multipart/x-mixed-replace` MIME type, a mechanism available in the Netscape browser for providing continuously updatable web pages. The previous work has used that for delivering stale data [9].

6 Evaluation

To measure the space and time efficiency of equivalence matching, we performed tests for the following three applications: map delivery with boundary error tolerance, information retrieval based on server-side image maps, and providing weather forecast based on zip code. We use real traces for the first two applications and a synthetic trace for weather forecast.

The goal of our evaluation was to estimate the performance improvement, if any, for DCCP-aware caching agents. For this, we mainly used two metrics: cache hit ratio and the volume of communication between the caching agent and the content provider (if this information is available). These metrics allow us to run repeatable experiments and to focus on the inherent performance improvements – independent of the network characteristics. In the ADL experiment, we also measured the end-to-end performance to demonstrate benefits in terms of response time reduction.

In these experiments, we ran a DCCP-aware proxy on a 248MHz, 128MB Sun Ultra-30 with Solaris 2.6. All the code was compiled with `gcc -O`.

6.1 Alexandria Digital Library web trace

In this experiment, we tried to estimate the performance improvement that can be achieved by using DCCP for spatial image searching and retrieval requests in the ADL system [3]. We have used an ADL user access trace for September and October 1997. This trace contains 69,337 requests and there are 28663 requests involving CGI. Of those dynamic requests, 21,545 were cgi-bin requests invoking one of two scripts: `draw_map` and `map-gif`. These scripts return portions of maps, based on the parameters which include central-latitude, central-longitude, height and width. Of those 21545 map requests, 7026 requests use an identical URL to access an ADL startup image with zero latitude and longitude and the remaining 14529 requests access maps with different parameter values. We used DCCP equivalence directives for those 14529 map requests to improve the cache hit ratio. We specified two map areas to be equivalent if the difference of above four parameters between two maps is within a specified error tolerance ratio (for example, 5% or 10%). Two requests are identical if the tolerance ratio is 0.

In this experiment, we set up a server at Rutgers University in New Jersey and ran both clients and the DCCP-enabled proxy at UCSB. The clients launch requests sequentially based on the trace to the proxy and the proxy can respond quickly if it has cached data since they are linked with a local network, or it fetches data from Rutgers. The round-trip latency between UCSB and Rutgers was around 82 milliseconds, measured by using "ping". The server at Rutgers does not have the ADL data installed, and it emulates the real ADL server at UCSB by executing each CGI request with processing time and result size same as what the UCSB ADL server would have. This experiment assumes that the server is fast enough to handle concurrent requests and it does not consider load impact among simultaneous requests. The experiment was conducted at midnight when Internet traffic is relatively stable and the average results are reported by running this experiment multiple times. There are minor variations among different runs; however, the deviation is fairly small.

The server processing time for each request was measured in 1997 [14]. At that time, we replayed the log against the ADL server to measure the response time and response size of each request. Considering today's machines can be 3-fold faster than the ADL server machine used in 1997, we have scaled down the request processing time accordingly in this experiment. The average processing time for all 28663 dynamic requests is about 0.5 seconds. The average processing time for all map requests (21545) is 169 ms while it is 162 ms for those 14529 requests whose results include the DCCP headers. The reason for the large difference in the average time between all dynamic requests and all map requests is that there are a certain number of ADL requests involving time-consuming spatial database searching.

Table 1 shows the benefits of equivalence caching for processing all dynamic ADL requests in this trace. The table shows that for exact matches, a 38.6% cache hit is achievable and this is because there are a large number of repeated requests with identical URLs. If the application can tolerate a 5% or 10% relative error, the hit percentage increases to 57.1% and 64.5% respectively. The total size

Error tolerance	0%	5%	10%
Number of cache hits	11074	16370	18482
Cache hit percentage	38.6%	57.1%	64.5%
Saved communication	48MB	65.2MB	75MB
Percent reduction in communication	41.5%	56.3%	64.8%
Ave. response time for a cache hit	4.8 ms	4.5 ms	4.4 ms
Ave. response time for a cache miss	773 ms	852 ms	958 ms
Ave. response time per request	476 ms	368 ms	340 ms

Table 1: Impact of result equivalence on cache performance when all dynamic requests are considered.

of data shipped in this trace is 115.6MB and the forth and fifth rows show the server-proxy communication volume saved due to the deployment of DCCP. The sixth row is the average request response time (from a client launches a request until it receives a result) for those requests whose results are fetched from the proxy cache at UCSB. The seventh row is the average request response time for those requests whose results are fetched from the Rutgers server due to cache misses. The eighth row is the average request response time for all dynamic requests. The results show that DCCP with tolerance 10% is 40% faster than only caching results of identical requests in terms of the average response time. If tolerance 5% is used instead, then it is 29.3% faster.

Now we focus on those map requests whose return results use the DCCP directives and there are 14529 such requests. Table 2 shows the benefits of equivalence caching and cost incurred due to the use of DCCP in processing those map requests. The table shows that for exact matches, only a 13.6% cache hit is achievable. If the application can tolerate a 5% or 10% relative error, the hit percentage increases to 49% and 62% respectively. The total size of data shipped for these requests is 61.3MB and the forth and fifth rows show the server-proxy communication volume saved due to the deployment of DCCP. The sixth row is the average request response time for all those map requests. The average cost for pattern matching network management and equivalence searching varies from 4.4 to 4.8 milliseconds per request while the memory usage for the entire pattern-matching network varies from 330KB to 900KB.

The performance improvement achieved by DCCP-aware caching agents comes, however, at the cost of additional memory to store the pattern-matching network. Even though the cost may not be large for many applications (e.g., see Table 2), it is conceivable that the memory requirement can grow rapidly if sufficient locality is not present in the request stream. This situation can be taken care of by imposing a bound on the amount of space used for the network. When the caching agent runs out of space for the pattern-matching network, it purged old entries using an LRU discipline. To understand the impact of such a restriction, we conducted additional experiments in which we limited the amount of memory that could be used for holding the pattern-matching network. Table 3 presents the results. The second row in Table 3 indicates percentage of space available compared to the situation when there is sufficient space. The third row shows the cache hit ratio under different space availability and error tolerance ratios. The result shows that this trace still has very good temporal locality and a high hit percentage can be maintained even when only half of the required space is available.

6.2 NASA Kennedy Space Center web trace

In this experiment, we tried to estimate the performance improvement that can be achieved by using DCCP for server-side image maps. For these experiments, we used a trace from the NASA Kennedy Space Center WWW server in Florida [1] to quantify the benefit of equivalence matching for image map type workloads. The trace contains

Error tolerance	0%	5%	10%
Number of cache hits	1975	7146	8959
Cache hit percentage	13.6%	49%	62%
Saved communication	7.9MB	26.MB	37MB
Percent reduction in communication	12.8%	46.6%	60.3%
Ave. response time per request	346 ms	217 ms	156 ms
Ave. cache search/insert overhead	4.8ms	4.5ms	4.4ms
Memory usage	900K	450K	330K

Table 2: Impact of result equivalence on cache performance for those DCCP-deployed requests.

Error tolerance	0%			5%			10%		
Percent space available	100%	80%	50%	100%	80%	50%	100%	80%	50%
Cache hit ratio	13.6%	12.8%	12.3%	49%	46%	45%	62%	58%	56%

Table 3: Impact of limited memory for the pattern-matching network.

1,569,898 requests, of which 20,925 are for the image map `countdown69`. We removed invalid requests, for a total of 20,774 requests. Image maps transmit the coordinates where the client clicked back to the server script, which translates the coordinates to a URL or returns a result. We have plotted the hit pattern in Figure 5. For each point that was clicked a hollow circle was plotted. The regions that have solid black areas are where multiple clicks occurred and correspond to buttons on the image map. Each button maps to the same URL, while points that do not fall into a button area result in an error message from the script. The image map `countdown69` has eleven buttons, as can be seen from the heavy concentration of dots in eleven regions. The scattering of dots outside of the buttons are clicks which do not correspond to buttons and result in an error message. We define each button as an equivalent region, as well as the error message as an equivalent region, resulting in twelve distinct regions on the image map.

Using standard URL matching results in a cache hit of 15,592 hits, and a 75.1% hit ratio. Using result equivalence for all points in the same region, the hit ratio increases to 99.9% or twelve cache misses, one for each region. The searching time and space cost is insignificant for this case because there are only twelve unique results and there are only twelve unique rules one per dynamic document.

It should be noted that for this trace, using client-site imagemaps can achieve the same goal as DCCP; nevertheless, this experiment demonstrates the effectiveness of DCCP if server-site images are deployed.

6.3 Weather forecast based on zip code

For this experiment, we generated a synthetic trace. We assume that the total number of zip codes is 99999 and this number is chosen based on the fact that a zip code in the U.S. has five digits. There are about 3143 counties in USA and for this test we assume that zip codes are randomly distributed among counties and each county in USA has the same weather condition within the same time period. In terms of access patterns of our synthetic trace, we assume a Zipf distribution [25, 8]. While we do not have evidence to support this assumption, we believe it is more realistic than a uniform distribution.

For a randomly generated trace with 500,000 requests, the average searching and network management cost is 0.12 milliseconds with a hit ratio of 99%. The reason for small time overhead is that there is a single application with one argument name (zip code). Hashing effectively indexes all zip codes, which allows for fast searching. The space usage is 1.6MB for indexing all 99999 zip codes in the pattern matching network because each rule enumerates all equivalent zip

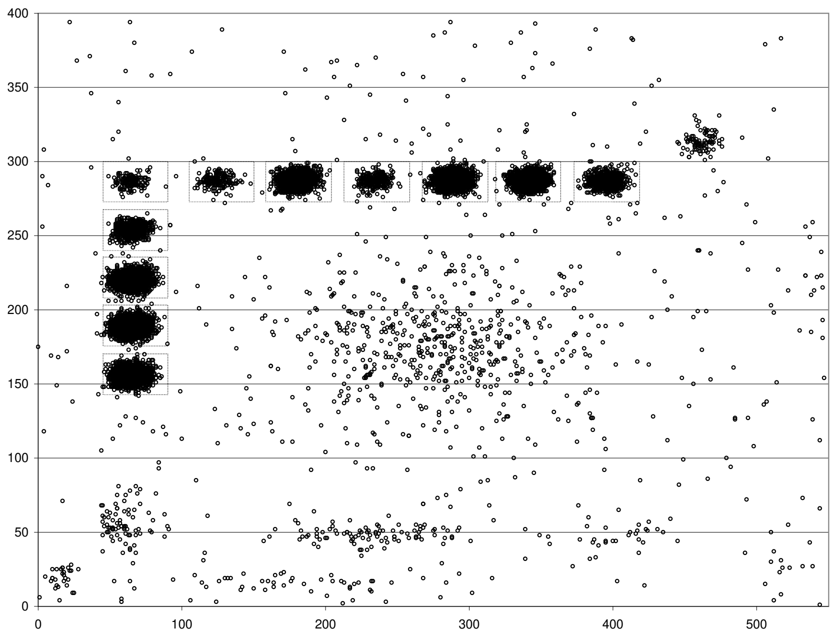

Figure 5: Hit pattern for `/cgi-bin/imagemap/countdown69` from the NASA trace. The hollow points correspond to coordinates where users clicked on the original image map. Boxes correspond to buttons on the original image.

codes for the corresponding result. Imposing a limit on space consumption does not degrade the cache hit ratio much because there are actually only 3143 unique results.

7 Concluding remarks

The primary contribution of this work is the DCCP protocol which allows a web application to specify result equivalence between the different documents (and groups of documents) it generates. This information can be used by proxies and other caching agents to speedup delivery of dynamic web content. We have illustrated the utility of this protocol for several applications and preliminary experiments indicate that the protocol is capable for achieving high cache hit ratios with fairly small space and time overhead.

DCCP can be extended to express more complicated patterns (e.g. inequality) and a few directives can be further added to the current DCCP for supporting basic features such as banner rotation and access logs, which are necessary for commercial Web sites [5]. Still, compared to Active Cache, the functionality of such a declarative protocol is more restrictive. The trade-off is that the declarative and lightweight nature of this caching protocol allows simplified security control and better efficiency.

We have not evaluated benefits of using our protocol for sending partially equivalent results and this issue needs to be addressed in the future. Under our current protocol implementation, searching equivalent results with complicated string matching rules can be time-consuming and caching may be less effective if imposing a searching time limit. One possibility is to restrict the use of string matching. Our current DCCP design is targeted at GET-based queries and there are a large number of dynamic requests which use POST-based queries. We plan to study the above issues after we gain more application experience with DCCP.

Acknowledgments. This work was supported in part by NSF CCR-9702640, IIS-9817432. We would like to thank Fred Douglis and anonymous referees for their detailed comments and thank K V Ravi Kanth and Ambuj Singh for providing their R* tree

code.

References

[1] ACM SIGCOMM. The Internet Traffic Archive. *http://ita.ee.lbl.gov/index.html*, Mar 1999.

[2] P. K. Agarwal and J. Erickson. Geometric range searching. In *Advances in Discrete and Computational Geometry (B. Chazelle, J. E. Goodman, and R. Pollack, editors)*, pages 1–56. Contemporary Mathematics 223, AMS Press, 1999.

[3] D. Andresen, L. Carver, R. Dolin, C. Fischer, J.Frew, M. Goodchild, O.Ibarra, R. Kothuri, M. Larsgaard, B. Manjunath, D. Nebert, J. Simpson, T. Smith, T. Yang, and Q. Zheng. The WWW Prototype of the Alexandria Digital Library. *Proceedings of ISDL'95: International Symposium on Digital Libraries*, Aug. 1995.

[4] G. Banga, F. Douglis, and M. Rabinovich. Optimistic deltas for WWW latency reduction. In *Proc. of USENIX'97*, pages 289–303, Jan. 1997.

[5] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching dynamic contents on the web. In *Proc. of Middleware'98*, pages 373–388, 1998.

[6] J. Challenger, P. Dantzig, and A. Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. In *Proc. of SC'98 (High Performance Networking and Computing. Formally known as SuperComputing)*, Nov. 1998.

[7] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic web data. In *Proc. of IEEE INFOCOM'99*, Mar. 1999.

[8] M. E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. In *Proc. of ACM SIGMETRICS Inter. Conf. on Measurement and Modeling of Computer Systems*, Apr. 1996.

[9] A. Dingle and T. Partl. Web cache coherence. In *Proc. of Fifth International WWW Conference*, May 1996.

[10] F. Douglis, A. Haro, and M. Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *Proc. of USENIX Symp. on Internet Technologies and Systems*, pages 83–94, Dec. 1997.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Network Working Group RFC 2616, June 1999.

[12] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, and P. Gautheir. Cluster-based network services. In *Proc. of 16th ACM Symposium on Operating System Principles (SOSP'97)*, pages 78–91, Oct. 1997.

[13] GeoSystems Global Corporation. The MapQuest Home Page. *http://www.mapquest.com*, 1999.

[14] V. Holmedahl, B. Smith, and T. Yang. Cooperative caching of dynamic content on a distributed web server. In *Proc. of Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC-7)*, pages 243–250, 1998.

[15] B. Housel and D. Lindquist. Webexpress: A system for optimizing web browsing in a wireless environment. In *Proc. of 2nd International Conf. on Mobile Computing and Networking*, pages 108–116, Nov. 1996.

[16] A. Iyengar and J. Challenger. Improving web server performance by caching dynamic data. In *Proc. of USENIX Symp. on Internet Technologies and Systems*, pages 49–60, Dec. 1997.

[17] Microsoft Corporation. The TerraServer Home Page. *http://www.terraserver.com*, Mar 1999.

[18] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proc. of SIGCOMM'97*, pages 181–94, Sept. 1997.

[19] D. Raggett, A. L. Hors, and I. J. (Eds.). HTML 4.0 Specification. W3C Recommendation, http://www.w3.org/TR/REC-html40/, April 1998.

[20] T. Smith. A digital library for geographically referenced materials. *IEEE Computer*, 29(5):54–60, 1996.

[21] A. van Hoff, J. Giannandrea, M. Hapner, S. Carter, and M. Medin. The HTTP Distribution and Replication Protocol. Submitted to W3C. http://www.w3.org/TR/NOTE-drp, August 1997.

[22] L. Wall, T. Christiansen, and R. L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., Sebastapol, California, second edition, Sep 1996.

[23] Xerox PARC. The Xerox Map Viewer. *http://pubweb.parc.xerox.com/map*, Mar 1999.

[24] H. Zhu, B. Smith, and T. Yang. Scheduling optimization for resource-intensive web requests on server clusters. In *Proc. of 11th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'99)*, pages 13–22, June 1999. http://www.cs.ucsb.edu/research/rcgi.

[25] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Cambridge, MA, 1949.