SEMANTIC AND COOPERATIVE DOCUMENT DELIVERY OVER DISTRIBUTED SYSTEMS

David Coquil and Harald Kosch
Department of distributed information systems
University of Passau, Germany
email:{david.coquil,harald.kosch}@uni-passau.de

Lionel Brunie
LIRIS
INSA de Lyon, France
email:lionel.brunie@insa-lyon.fr

ABSTRACT
Modern large-scale distributed information systems need to implement specific techniques in order to favor the efficient processing of document access requests. This paper proposes a delivery system that consistently integrates two types of such techniques based on caching: cooperative caching, which promotes the exchange of documents between caches whenever advantageous, and semantic caching, which uses a monitoring of popular topics in order to take decisions on which documents to cache. It also presents the results of the experimental evaluation of this proposal that has been conducted thanks to a simulator of the complete system; these experiments show the potential of the system for effectively reducing access request processing times.

KEY WORDS
distributed system, semantic caching, document retrieval, cooperative caching, simulation

1 Motivation and related work

A typical use case that is common to all modern distributed information systems in the broader sense of the term (Web/Web 2.0, datagrids, peer-to-peer systems…) is that of a document with an initially unique source that is accessed in a short time interval by several clients distributed over a very large geographical area. As this scenario becomes increasingly inefficient with the number of clients, all these systems implement some form of data dissemination control, so that most clients access a closer copy of the requested resource instead of querying the original source; this is typically achieved with *caching techniques*. However, without coordination, the use of these resources can still be sub-optimal: for example, a client may not be aware of the existence of a copy in a nearby cache, or a document may be unnecessarily duplicated in several close caches. Such inefficient management results in a waste of network bandwidth, and in unnecessarily long retrieval times. *Cooperative caching* techniques can help organize the use of caching resources of the global system in a rational and efficient way.

In addition, an efficient cache management policy can be seen as a form of prediction of future user requests. Traditional caching heuristics rely on operational parameters (last access date, document size, access frequency…) to achieve these predictions. However, user interest for specific documents chosen from a large set of documents is not purely random, but is partially correlated to their subjects, their semantic information. Trends in interest for specific subjects can be observed in requests distributions ([1]). In parallel, semantic information is increasingly directly attached to documents that are made available in a distributed information system; this is the case for example with semantic Web applications and Web 2.0 folksonomies. This makes it easier to monitor the distribution user requests with respect to this information. *Semantic caching* techniques can then be implemented in order to take advantage of this knowledge by keeping interesting documents in the caches and by pre-fetching known interesting documents that are no longer in cache.

Although there have already been a number of semantic caching proposals (see [2, 3] for an overview, [4] for peer-to-peer networks, [5] for mobile environments), they usually amount to storing query results in order to avoid computing the same complex queries several times; these proposals are not aimed at improving the processing of requests of access to documents for which semantic information is already known. In a related area, many researchers have made contributions towards the development of improving the quantity and quality of semantic information available about documents that are available in distributed systems: these are the "semantic Web" ([6]) and more recently "semantic Grid" ([7]) areas of research. While many interesting applications have been derived from these works, their approach is overkill at the level of a document cache. Their complexity does not fit an environment in which decisions have to be taken as fast as possible about what to keep, discard or pre-fetch if the whole purpose of caching (improving retrieval efficiency) is to be fulfilled. Moreover, caches do not need the level of details offered by these approaches: an identification of the main and possibly secondary topic(s) related to the document constitutes sufficient semantic information for their purpose, hence the relevance of works in the area of automatic document classification in our context, such as [8] for the specific case of Web pages.

There has also been a number of cooperative caching proposals in various environments: in the Web ([9]), grids ([10]), mobile ad hoc networks ([11]), content delivery networks ([12]); however, they do not consider the possibility of semantic caching. While in theory nothing prevents both types of techniques to be used at the same time, an effective joint implementation can be problematic. Indeed, both types of techniques make decisions about moving, keeping, deleting and pre-fetching documents; these decisions can be inconsistent, effectively nullifying each othert's effects. In this paper, we propose a complete document delivery system implementing both cooperation mechanisms and usage monitoring-based semantic caching techniques that have been designed from the start to be compatible.

The rest of this paper is organized as follows. Section 2 describes our semantic caching proposals. Section 3 outlines our proposed cooperative architecture that also integrates the semantic caching mechanisms. The results of the simulation-based experimental evaluation of the system is outlined in section 4. Finally, conclusions are drawn in section 5.

2 Semantic caching

The principle of our semantic caching proposal is to consider recent user requests as hints of current user interests. This means that we postulate that if several requests related to a specific subject are observed, the probability of request of all documents related to the subject increases. We extend this idea to different subjects: an influx of requests for a subject increases the probability of requests for another semantically related subject. To make use of this hypothesis in a real-world environment, the "strength" of the relation needs to be considered: tenth century chants and the latest popstar may be related as part of the general "music" concept, but the dynamic of their requests is probably very different. This can be observed by analyzing requests logs.

Our goal is to translate these assumptions into cache management heuristics. To this end, we have defined a data structure for storing the semantic information related to a set of documents and the relations between the concepts indexing them. This data structure, shown in figure 1, is a single-rooted n-ary tree. Its leaves represent documents, while its internal nodes correspond to tags or keywords. The edges between a tag and a document correspond to an indexing relation, while those linking internal nodes correspond to a generalization/specialization relation. The *weight* associated to the edges evaluates the strength of the relation between the linked nodes as described above. It is computed through an off-line analysis of request logs. The corresponding algorithm is conditional-probability-based: the weight of a A-B link is evaluated as the probability of a request for a document directly linked B in the hierarchy given a request for a document linked to A (the B query has to occur in a given time interval after the A query). The algorithm has $O(n^2)$ time complexity with respect to the size of the logs used for the calculation.

Based on this data structure, we have defined the *temperature*, a numerical value attached to each document meant as an instant measurement of the likelihood that it

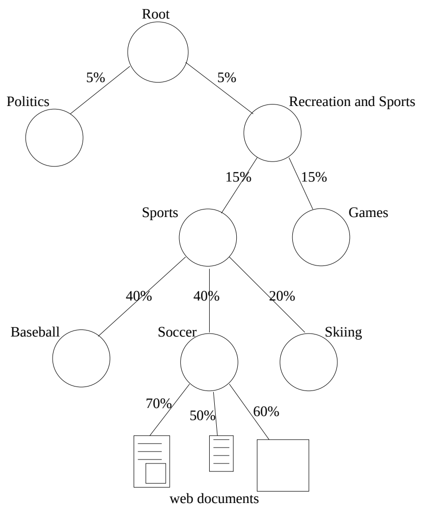

Figure 1. Graphical representation of the data structure

will be accessed in the near future. This value is computed through a synthesis between the number of recent access requests for the document and the semantic links of the data structure.

The temperature computation process is performed at regular intervals. The number of accesses to each document between two consecutive computations is stored. The process is carried out in three phases which we will now describe using the example outlined in figure 2. The nodes depicted in the figure are to be understood as being part of a larger hierarchy. In phase 1, the temperature of documents is increased by their number of accesses since the last update; this is the case for documents D_1 and D_3. The temperature of documents that were not accessed is decreased by a value equal to their latest temperature increase (if any), hence the decrement of 2 for D_2's temperature. In phase 2, the temperature variations of the documents are recursively propagated upstream, following the links of the data structure. The propagated values are always multiplied by the weight of the edges through which they are passing. Thus for example N_1 receives $\Delta(D_1) * W(D_1, N_1) + \Delta(D_2) * W(D_2, N_2)$. Finally in phase 3, the temperature variations resulting from the previous phase are propagated downstream proportionately to the weights until they reach the documents. In figure 2, the initial temperature variations noted for N_1, N_2 and N_3 are supposed to originate from other non depicted documents, hence the difference with the values noted for phase 2.

Temperature serves as the basis of the COldest DOcument Replacement (CODOR) cache replacement policy: when a cache is full and a new document must be inserted, the documents with the lowest temperature are recursively deleted until enough storage space is freed. Temperature can also be used to pre-fetch documents that are already known but are no longer in the cache and that have a temperature at least higher than that of the "coldest" document currently present in the cache.

The temperature computation algorithm presented above can be implemented in lienar time with respect to

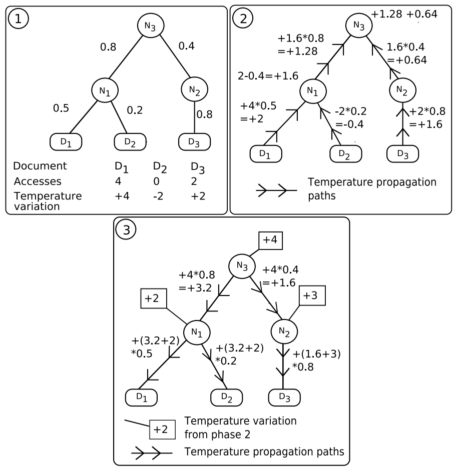

Figure 2. Three-phase temperature computation process

the number of documents indexed in the data structure. The CODOR algorithm has a complexity of quadratic order $O(n^2)$, with n the number of documents stored in the cache at the time of the replacement decision.

3 Cooperation

Our cooperative policy for the processing of document access requests is based on a three-level hierarchy of proxy servers, the third level being optional (see figure 3).

The first level component, the Local Proxy (LP), corresponds to a cache attached to a client. In the Web framework, its position corresponds to that of the caches included in browsers. However, instead of being isolated, it is designed to be able to cooperate with other nearby LPs in order to efficiently process the requests of its users, with which it can exchange documents according to a mechanism synchronized by higher-level proxies.

The second-level component, the Intermediate Proxy (IP), is similar to a Web proxy server. It implements the CODOR and pre-fetching policies described in the preceding section. Through the Meta Proxy described below, the IPs can exchange documents with each other and with LPs. They monitor temperature variations and exchange information about the most noteworthy variations with other IPs. Each LP is compulsorily linked to a single IP. The IP receives the cache misses of its LPs, possibly indicating them that a cached copy of the requested document can be fetched from another LP. To this end, it maintains an index of the contents of the cache of its LPs.

The highest level component, the Meta Proxy defines a global view of several IPs with their sets of LPs. No data is actually stored at this level; instead, an MP maintains an aggregation of information about the caching hierarchy that it is responsible for. In particular, it serves as a virtual directory of the content cached in its hierarchy. It also exchanges this information with other MPs on a peer-to-peer mode, provided that they are close enough for cooperation with them to be worthwhile. An MP thus knows about the contents of a large number of caches. It uses this knowl-

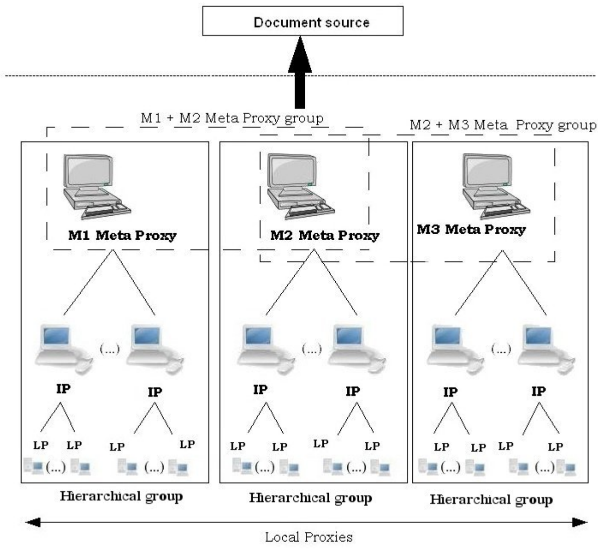

Figure 3. Cooperative architecture

edge to implement the second level of the cooperative policy: on an IP cache miss, it may tell it about a cache from which the requested document can be fetched. Each IP is compulsorily linked to an MP. This level of the architecture is optional; in its absence, its functions are ensured by the IPs.

This architecture is structured by two types of *virtual groups*:

- **Hierarchy groups**, which group together the proxies that are part of the same Meta Proxy $\Rightarrow$ Intermediate Proxies $\Rightarrow$ Local Proxy hierarchy.
- **Meta Proxy groups**, which group together several cooperating MPs, and therefore several hierarchy groups. These groups have the advantage of increasing the level of cooperation. For example in figure 3, thanks to the M1+M2 and M2+M3 groups, the M1 and M3 hierarchies indirectly benefit of each other's cache contents even though they do not directly interact.

For creating the cache context indexes at the IP and MP level, the Cache Digest protocol ([13]) is used. Based on Bloom filters, it allows to maintain such information up to date using a messages system while limiting the overhead created by its use. Cache Digest messages are also exchanged by MPs belonging to the same MP group.

Listing 1 presents a summary of the query processing algorithm in the cooperative architecture. Note that some of the steps that seem consecutive can actually be performed simultaneously (such as searching for the document in the IP index and in its cache).

Listing 1. Document search algorithm

```
Client submits document access request
Request is transmitted to LP
Document is searched in LP cache
If it is found
```

```
 Document is transmitted to client
Else
 Request is transmitted to the IP
 Document is searched in the IP cache
 If it is found
  Document is transmitted to LP
  Document is copied in LP cache
  Document is transmitted to client
 Else
  Document is searched in the IP index
  If it is found in the index
   Document location is sent to the LP
   LP fetches document
   Document is copied in LP cache
   Document is transmitted to client
  Else
   Request is transmitted to the MP
   Document is searched in the MP index
   If it is found in the index
    Document location is sent to the IP
    IP fetches document
    Document is copied in IP cache
    Document is transmitted to LP
    Document is copied in LP cache
    Document is transmitted the client
   Else
    MP fetches document from its source
    Document is transmitted to IP
    Document is copied in IP cache
    Document is transmitted to LP
    Document is copied in LP cache
    Document is transmitted to client
   Endif
  Endif
 Endif
Endif
```

The possible interactions of the proxies of our system (document and information exchange) define the request-reponse based Temperature-Based Cache Control Protocol (TBCCP). TBCCP implements all the procedures defined in this section, with the exception of those related to the maintenance of the cache content indexes (responsibility of Cache Digest) and the direct transfer of files between proxies and/or sources (assumed to be a basic service offered by the distributed information system). The TBCCP messages are formed of a fixed-size header and a variable size body. The main field of the header defines the message type. The main message types are queries (PL to PI or PI to PL), with the following possible responses: HIT (the receiver has the document in its cache), FOUND (the receiver knows of a cache holding the document and forwards its address to the initial requester), NOTFOUND (the receiver neither holds the document nor knows where to find it in the system; the request is going to be forwarded), and AVAILABLE (the receiver retrieved the document from another source, the initial requester can now download it).

4 Experimental evaluation

4.1 Simulator

A simulator was developed in order to be used for the experimental evaluation of our proposals. The simulator enabled us to perform tests in the context of a environment including a number of clients, local proxies, intermediate proxies, content servers without having to deal with the constraints related to experimenting with such a complex system in a "real-life" environment.

The goal of the experiments was to assess the complete system with respect to the following aspects: the efficiency of the cooperation policy at the local proxy level, the efficiency of the cooperation policy at the intermediary proxy level and the efficiency of the temperature-based cache management policy within the complete system.

For the latter, we compared the performance of a simulated system using our temperature-based algorithm with those of the same system using three different other well-known replacement policies. Four different versions of the simulator were therefore developed, differing only in their replacement algorithm.

The simulators have been developed in Java, using the network simulation packages of the J-Sim simulation development environment (`http://www.j-sim.org`). J-Sim offers a number of functions that we needed for our simulator: state-based simulation, time reference framework, event ordering with respect to predefined trigger rules and duration, etc. The basic principle of the simulators is simple: they take a request distribution file and a system configuration file (defining the clients, LPs, IPs, and the links between them) and process the requests until the end of the request distribution file, while updating the requested measures.

4.2 Query distribution

We have used logs collected on real-world Web proxies for our experiments; these data have been obtained from the IRCache project (`http://www.ircache.net`). The files come from various mostly academic Web proxies. IP addresses contained in the files are anonymized, but in such a way that it remains possible to determine which requests correspond to the same client. However, the anonymization process is not consistent between different days; meaning that the experiments could only be conducted over logs covering at most 24h traffic. This explains the relatively low value of some of the caracteristics outlined in table 1.

Table 1. Log files caracteristics

Average number of requests	127224
Average object size (Ko)	17,2
Total size served (Go)	2.19
Average number of clients	125
Maximum possible hit rate	52,3

Some pre-processing on these files was necessary. Currently unavailable URLs for which GET requests return HTTP 404 or other types of error had to be identified and eliminated. Moreover, as our system needs semantic information that is not yet attached to current Web documents, the available documents were indexed off-line.

4.3 Experimental results

4.3.1 Cooperation between local proxies

The goal of this experiment is to assess the impact of local proxy cooperation. The simulated system is composed of one Intermediary Proxy and as many local proxies (with identical cache sizes) as the number of distinct clients found in the log file. Figure 4 presents the evolution of the hit rate of the IP and of the LPs with respect to the size of a local cache. One hit for the LPs means that the IP has found a requested file in the cache of another LP.

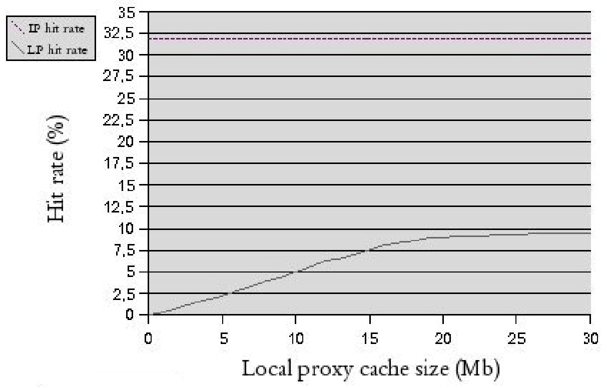

Figure 4. Hit rate with respect to local proxy cache size

The IP hit rate remains constant, as the IP always receives the same requests regardless of the size of the LP caches. The LP hit rates increases regularly before saturating around 20 Mb. This is due to the fact that the average total size of the fetched files for one client in the log files is close to 20 Mb.

4.3.2 Cooperation between intermediary proxies

This experiment simulates a 2-IP system, each one with its own hierarchy of LPs. The measured value here is the *cooperation rate*, meaning the ratio of the number of requests that one IP could resolve by fetching the document from a cache of the other hierarchy to the total number of requests. Figure 5 outlines the evolution of this value with respect to the cache size of the IPs.

The rate first increases in a fast and quasi-linear way, before the curve starts to bend around 500 Mb. For caches larger than 1.4 Gb, the rate is quasi-constant. We conclude from this experiment that the IP cooperation policy alone

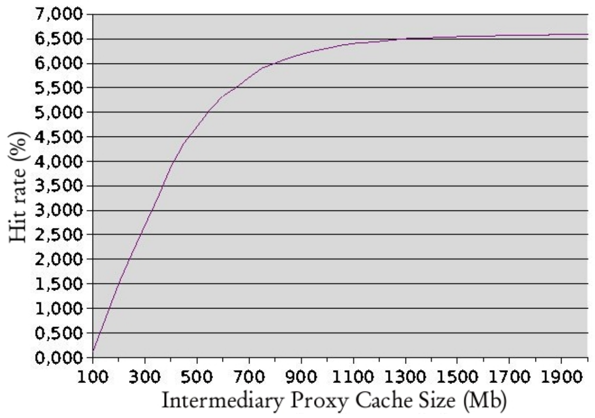

Figure 5. Cooperation rate with respect to IP cache size

allows to improve the processing of around 7% of queries, which is far from negligible.

4.3.3 Replacement policy

This experiment compares the efficiency of a 2-IP system implement IP cooperation, LP cooperation and temperature-based replacement with that of the same system with a different cache replacement policy. The selected adversaries are Least Recently Used (LRU), Least Frequency Used (LFU) and Greedy Dual Size (GDS). LRU and LFU were chosen because they are based on the two operational parameters that are the most significant in Web requests distributions (respectively frequency of access and last access date), while GDS has been several times found to be efficient in Web caching settings ([14]). The results for all four algorithms are displayed in figure 6.

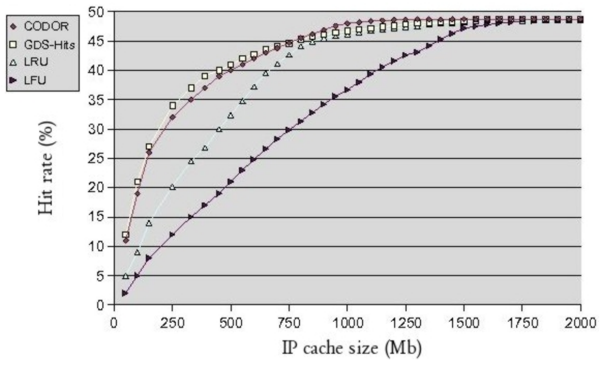

Figure 6. Hit rates for different replacement policies

The figure shows a very similar evolution for CODOR and GDS, with the difference between them never exceeding two points. GDS is superior for smaller cache sizes, however CODOR's bends less heavily when the size increases; CODOR is the algorithm that reaches its maximum performance the first. The difference between these algorithms and the two others is bigger for smaller cache sizes, especially with respect to LFU.

To conclude on these experiments, they show that the three potential improvement factors of our system do indeed all have a positive impact on the performances. Moreover, two of the three experiments were conducted in a complete system environment, which proves that the cooperation and semantic are conducted simultaneously and consistently, not nullifying each other's positive effects.

5 Conclusion

In this article, we have presented an architecture for the efficient delivery of documents in a distributed system that integrates semantic caching and cooperation mechanisms in a consistent way. Our semantic caching proposals are based on the monitoring of previous user requests, on a limited knowledge of the semantic properties of the documents in the form of keywords, and on a knowledge of the semantic links between some of the keywords. We compute a synthesis of these properties, the temperature, that aims at being a metric of the interest of a given document for a cache. Our cooperative architecture implements document sharing between caches at a local and global level, allowing caches to fetch documents from other nearby caches rather than from the distant source. It also implements the sharing of information by monitoring temperature variations. We have performed an experimental evaluation of our system using simulation, and have found out the three potential factors of improvement of the integrated system (local proxy cooperation, intermediate proxy cooperation, semantic caching) do indeed all have a positive impact on the efficiency of requests processing.

Future work in this domain includes additional experiments, such as the study of the effect of the complexity of the architecture on performances (scalability of the architecture, evaluation of the limitation to the number of cooperating proxies above which the system does not benefit from cooperation anymore). We also plan on studying the influence of the accuracy of the semantic information that we use: while in its current state our system treats this information as external and accurate, collectively created Web2.0 folksonomies have been praised as much more relevant than traditional keywords determined by Web sites authors, and it would be interesting to see if our system can indeed benefit from this property.

References

[1] E. Adar, D. Weld, B. Bershad, & S. Gribble, Why we search: visualizing and predicting user behavior, *Proc. 16th Int. World Wide Web Conf.*, Bank, Canada, May 2007.

[2] Q. Ren, M. Dunham, & V. Kumar, Semantic caching and query processing, *IEEE Transactions on Knowledge and Data Engineering*, *15*(1), Jan./Feb. 2003, 192–210.

[3] B. Jónsson, M. Arinbjarnar, B. Pórsson, M. Franklin, & D. Srivastava, Performance and overhead of semantic cache management, *ACM Transactions on Internet Technology*, *6*(3), 2006, 302–331.

[4] P. Garbacki, D. Epema, & M. Van Steen, A two-level semantic caching scheme for super-peer networks, *Proc. 10th IEEE Int. Workshop on Web Content Caching and Distribution*, Sophia Antipolis, France, 2005.

[5] B. Zheng, W. Lee, & D. Lee, On semantic caching and query scheduling for mobile nearest-neighbor search, *Wireless Networks*, *10*(6), Nov. 2004, 653–664.

[6] N. Shaboldt, W. Hall, & T. Berners-Lee, The semantic Web revisited, *IEEE Intelligent Systems*, *21*(3), 2006, 96–101.

[7] D. De Roure, J. Frey, D. Michaelides, & K. Page, The collaborative semantic grid, *Proc. 2006 Int. Symposium on Collaborative Technologies and Systems*, Las Vegas, Nevada, USA, 2006.

[8] X. Qi & B. Davison, Knowing a Web page by the company it keeps, *Proc. 15th ACM Int. Conf. on Information and Knowledge Management*, Arlington, Virginia, USA, 2006.

[9] A. Wolman, M. Voelker, N. Sharma, & N. Cardwel et al., On the scale and performance of cooperative Web proxy caching, *Proc. 17th ACM Symposium on Operating Systems Principles*, 16–31, Charleston, South Carolina, USA, 1999.

[10] Y. Cardenas, J. Pierson, & L. Brunie, Management of a cooperative cache with grid cache services, *Concurrency and Computation: Practice and Experience*, *19*(16), 2007, 2141–2155.

[11] L. Yin & G. Cao, Supporting cooperative caching in ad hoc networks, *IEEE Transactions on Mobile Computing*, *5*(1), Jan. 2006, 77–89.

[12] Jian Ni & D. Tsan, Large-scale cooperative caching and application-level multicast in multimedia content delivery networks, *IEEE Communications Magazine*, *43*(5), May 2005, 98–105.

[13] A. Rousskov & D. Wessels, Cache digests, *Computer Networks*, *30*(22-23), 1998, 2155–2168.

[14] C. Lindemann & O. Waldhorst, Evaluating the impact of different document types on the performance of web cache replacement schemes, *Proc. Int. Conf. on Dependable Systems and Networks*, Bethesda, Maryland, USA, 2002.