MultiCache: an incrementally deployable overlay architecture for information-centric networking

Konstantinos Katsaros, George Xylomenos and George C. Polyzos
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece
E-mail: ntinos@aueb.gr, xgeorge@aueb.gr and polyzos@aueb.gr

***Abstract*— It has been long realized that the Internet is evolving from a network connecting pairs of end hosts to a substrate for information dissemination. While this shift towards information centric networking has been clearly demonstrated by the proliferation of file sharing (e.g., BitTorrent) and content delivery (e.g., YouTube) applications, it has not been followed by a corresponding shift in network architecture. As a result, even though such applications are attractive to both content providers, due to their lower bandwidth requirements, and to end users, due to their reduced download times, they plague the underlying network with redundant packet transmissions, a significant part of which takes place over costly inter-domain links. In essence, the end-to-end nature of the current Internet architecture prevents network operators from controlling the traffic carried by their networks, delegating such control to end users and their applications. In this paper, we propose MultiCache, an information centric architecture aiming at the efficient use of network resources that is based on two primitives: multicast and caching. To this end, we revisit overlay multicast as a means for content delivery, and take advantage of multicast forwarding information to locate, in an anycast fashion, nearby caches that have been themselves fed by multicast sessions. Our architecture is evaluated against a widespread file sharing application (BitTorrent) with respect to both network resource savings and end user experience.**

I. INTRODUCTION

It has been long realized that the Internet's communication model does not reflect current end user usage patterns. While users focus on the desired information, the underlying communication substrate focuses on the end-to-end communication between pairs of end-hosts. Inevitably, a translation between the information domain and the networking domain takes place, typically consisting of the establishment of a delivery path between the data provider and the data consumer. This translation is usually performed inefficiently as it is based on end-point centric overlay data delivery structures that neglect network topology, data location and data popularity, ultimately over-consuming network resources. In the characteristic example of P2P file sharing applications, it has been shown that a major part of the incurred traffic crosses Internet Service Provider boundaries, even though the corresponding information could have been retrieved locally [1].

We believe that at the heart of this problem lies the lack of information awareness inside the network, that is, the fact that only the end-points are aware of *what* is being delivered. Due to this deficiency, attempts to more efficiently use network resources, such as resource sharing via caching and multicast, are hard to succeed, as the corresponding decisions are made at the end-points of the network, based on coarse grained information. Therefore, an information-centric model should be employed in order to enable the efficient use of network resources and better reflect user needs. In this context, the network, by gaining knowledge on *what* is being delivered, in addition to *where* it originates from or is destined to, becomes inherently capable of forming targeted and efficient delivery structures. In addition, in this paradigm users only express their interest on pieces of information, rather than engaging in the aforementioned mapping between models.

Towards this direction, we present *MultiCache*, an overlay network architecture that brings information into focus. MultiCache aims at taking advantage of information-awareness to improve the utilization of network resources via resource sharing. To this end, network operators deploy and control proxy overlay routers that enable the joint provision of multicast and caching, targeting both synchronous and asynchronous requests. End hosts interact with the infrastructure by simply providing flat, location independent identifiers for the desired content, without engaging in the process of locating an end host providing the data. Inside the network, the Scribe overlay multicast scheme [2] is employed to transport the content from its origin in a publish/subscribe fashion, thus serving synchronous requests (e.g., flash crowds) and feeding in-network shared caches. By taking advantage of the locality awareness of the established Pastry routing substrate [3], anycast queries based on the already established overlay multicast forwarding state are later used to locate nearby caches that can serve asynchronous requests by unicasting the cached content.

In this paper, we provide a thorough description of the proposed architecture, highlighting its primary design choices. Moreover, we gain insight on its performance by comparing a simple MultiCache-based content distribution application with the prevalent BitTorrent file sharing application. Preliminary simulation results demonstrate the potential of MultiCache to better utilize network resources, while yielding improved download times for the end users. The remainder of this paper is organized as follows. In Section II we describe the proposed architecture, providing a performance evaluation in Section III. In Section IV we present previous research relating to our

work. We describe our next steps and conclude in Section V.

II. Proposed Architecture

A. Deployment

MultiCache functionality is deployed in an overlay fashion inside access networks. This entails the deployment of additional infrastructure in the form of *Overlay Access Routers* (OARs), possibly collocated with regular access routers. OARs provide the following functionality:

1) They participate in the overlay routing and forwarding substrate, enabling the use of overlay multicast. This entails the maintenance of Pastry routing information [3], as well as the functionalities of Scribe and MultiCache, as described below.
2) They act as proxies of end-hosts in the overlay, i.e., an end-host establishes a control connection to an available OAR designated during network attachment. The selected OAR (*proxy* OAR) may be collocated with the access router of the end-host or it may be located several hops away, subject to the density of OAR deployment. The role of the proxy OAR is to act as the interface of the end-host to the overlay, possibly aggregating data requests from multiple attached end-hosts.
3) They cache content destined to their attached end hosts. As a result, the same content is cached at multiple locations in the network, i.e., at all leafs of an established overlay multicast trees.
4) They provide cached content to other OARs via unicast, as described in Section II-C.

The deployment of overlay functionality inside access networks serves several important goals. First, the overlay character of the architecture facilitates the deployment process, as it does not require the replacement of existing infrastructure, while it allows the unobstructed operation of established services and applications. By deploying MultiCache inside access networks, content is cached close to the clients [4], facilitating the discovery of caches in the clients' networking vicinity and therefore enabling the localization of traffic (see Sections II-C.2 and III). Finally, as discussed in [5], placing caches close to the end points of the network avoids incentive incompatibilities regarding inter-domain relationships. A simple deployment example is given in Figure 1, with OARs being collocated with the corresponding access routers.

B. Multicast

Multicast forwarding takes place among OARs driven by end-host requests, i.e. after end-hosts issue requests for desired data objects to their proxy OARs via the established control connections. These requests may be translated to corresponding Scribe JOIN messages, depending on the current state of the proxy OAR with respect to the indicated data item. The joining process deviates slightly from regular Scribe in that JOIN messages are extended to further carry the IP address, the listening port number, the credentials of the initial issuer of the JOIN message (i.e. the proxy OAR[1]) and the 32-bit

[1]Note that in cases of multi-overlay hop paths, the proxy OAR is not the node that eventually delivers the JOIN message to an already joined node.

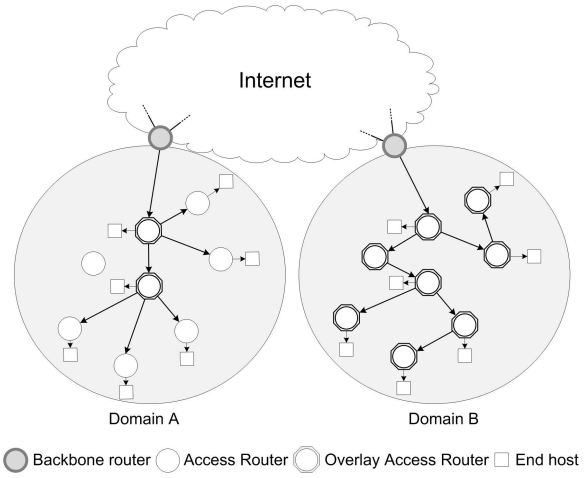

Fig. 1. MultiCache Deployment

Autonomous System (AS) number of the proxy OAR's AS. This extra information is used during cache searching and provisioning, as explained in the next subsection.

During the joining process, OARs establish TCP connections with their children for the reliable delivery of the requested data. When a JOIN message eventually reaches the *Rendez Vous* (RV) point, the content provider will be solicited to deliver the data which will then start traversing the tree created via the already established TCP connections. It is assumed that the content provider has already created the respective group, and therefore has contacted the RV point. Due to the asynchronous character of request arrivals, this process may result in partial data availability at the leafs of the multicast tree at the end of the multicast session. However, the caching mechanism ensures that these partial feeds will be able to complete later.

A simple example of these operations is given in Figure 2. The first two subfigures depict progressive snapshots of a simple Scribe tree. The arrays below each OAR denote the availability of the content and will be further explained in Section II-D. OAR *1* first joins a Scribe multicast tree via OAR *2*, followed by OARs *3* and then *5* that join during the multicast session via OAR *4*. At the end of the multicast session (end of Step 3) , OARs *3* and *5* have received part only of the multicasted content, i.e., they are missing blocks 0 to 3 and 0 to 6 respectively.

C. Caching

1) Protocol description: In order to locate an available cache, MultiCache exploits the already established multicast forwarding state. For this reason, forwarding nodes cache the forwarding state established during tree creation even past the end of the multicast session. As caches are created, CACHE UPDATE messages are issued by leaf OARs towards the RV of the multicast tree. The purpose of these messages is to notify ancestors about the availability of cached items and thus allow their discovery upon a cache request. Note that caches may be fed by other caches, therefore OARs cannot rely solely on forwarded traffic to gain knowledge of downstream cache

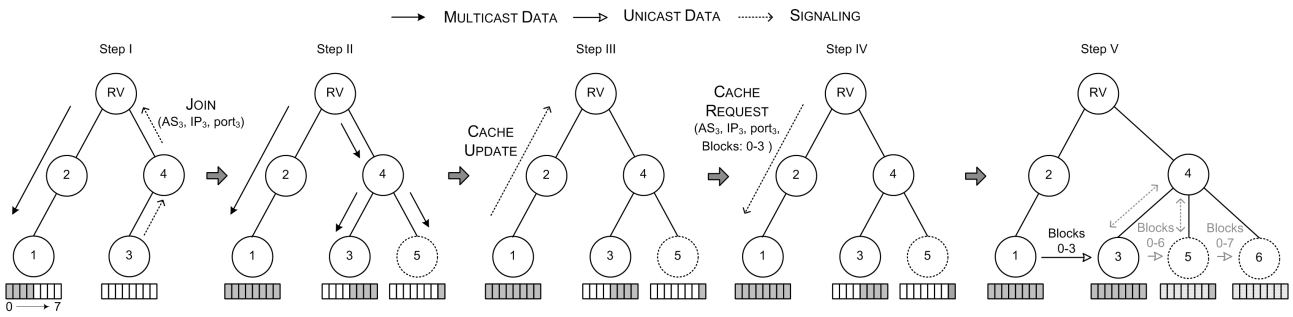

Fig. 2. MultiCache example

existence. OARs further propagate received CACHE UPDATE messages towards the root *iff* they have not already done so for another descendant cache, thus avoiding feedback implosion.

Following regular Scribe operation, a Scribe JOIN message is suppressed at the first OAR that has already joined the respective tree, henceforth termed as a *meta-cache* OAR. Depending on the arrival time of the JOIN message and its current state with respect the indicated object, the meta-cache OAR may issue a cache request (CACHE REQUEST) message on behalf of the joining proxy OAR. This message carries all the extra information inserted in the JOIN message. Obviously, if the meta-cache OAR happens to have cached the requested object, the cached data will be directly delivered to the requesting node. Otherwise, if the meta-cache OAR has completed forwarding the data object to its descendants[2], it will anycast a cache request (CACHE REQUEST) message to its downstream sub-tree in a *depth first search* (DFS) fashion. In particular, at each level of the traversed sub-tree the request message is forwarded to one of the children that have previously issued a CACHE UPDATE message. At each step, preference is given to children belonging to the same AS with the proxy OAR that will be eventually served. Among equivalent children candidates, a randomized selection ensures the uniform distribution of load to the available caches.

If the meta-cache OAR is currently forwarding multicast data, it forwards the remainder of the arriving multicast data to the joining node, as described above, keeping also track of the part of the data object that was not delivered due to the late arrival of the JOIN message. This accounting mechanism will enable the precise indication of the required cached content in later cache requests. Upon the arrival of the first CACHE UPDATE notification, a CACHE REQUEST is issued for each partially served child of a meta-cache OAR. Eventually, a CACHE REQUEST reaches an OAR in the CACHED state and a TCP connection is established between the caching OAR and the proxy OAR for the delivery of the cached data.

In the example of Figure 2, due to the arrival of OAR *3*, the RV node receives a JOIN message from OAR *4* including OAR *3*'s details (Step I). At the end of the multicast session OAR *1* completes downloading the data and notifies its parent with a CACHE UPDATE message (Step III). OAR *2* forwards this notification to the RV point, where it gets suppressed. At this point, having partially served its descendants, the RV node issues a CACHE REQUEST towards the only child known to lead to a cache, i.e. OAR *2* (Step IV). Since this node has not cached the requested item, it forwards the received message to OAR *1* which eventually serves the request (Step V). The same procedure takes place in the case of OARs *5* and *6*, which belong to a different AS than OARs *3* and *4*. OAR *5* joins first and receives the cached data from OAR *3*. Later, OAR *6*'s JOIN message is suppressed by OAR *4* that prefers to send a CACHE REQUEST message to OAR *5* than to OAR *3*, resulting in localized traffic between OARs *5* and *6*.

2) Locality properties: The expectation of localized cache hits builds on Pastry's properties and the multiplicity of cache locations. According to Pastry's route convergence property [3], since caching OARs are essentially leafs of the data item's Scribe multicast tree, a Scribe JOIN message of an arriving proxy OAR is expected to reach a meta-cache OAR at a distance approximately equal to the distance to a caching OAR in the proximity space. At the same time, following Pastry's prefix based routing, Scribe JOIN messages are initially expected to travel short distances at each overlay routing step. Hence, as demonstrated in [6], in cases of multiple cache locations, Scribe JOIN messages of arriving proxy OARs are expected to first reach meta-cache OARs leading to nearby caches. In effect, cache search messages and cached data are expected to traverse short network distances, with respect to Pastry's proximity metric, leading to the localization of traffic. This is further assisted by the simple AS number-based cache selection mechanism.

3) Cache size and expiration: One major issue pertaining to every caching scheme is the cache replacement policy. In this stage of our research, we attempt to establish a performance baseline and investigate the potential benefits of the proposed architecture, therefore we assume that we have infinite cache capacity. Our plans for imminent work focus on this issue. Specifically, our plans consider connecting cache availability with the existence of the corresponding overlay multicast forwarding state. In this context, we consider caching OARs evicting items from their cache only after they have left the corresponding multicast tree. The whole procedure will be initiated based on the selected cache replacement policy (e.g., LRU, LFU). Note that cache invalidation due to content updates is implicitly addressed by the proper selection of new flat identifiers at the application layer.

[2]The anticipation of this fact is based on Content Fragmentation (see Section II-D) and a simple block counting mechanism.

D. Content fragmentation

MultiCache allows the fragmentation of large files into *pieces*, as in BitTorrent, leading to the creation of a forest of Scribe trees as in [7]. This fragmentation serves several important goals. First, it facilitates the establishment of parallel data flows towards a recipient node, possibly better exploiting the available downlink bandwidth and avoiding the sequential delivery of large files. Moreover, it allows the partial caching of large data volumes, i.e. certain pieces can be cached independently of others, enabling a fine grained management of caching space [8]. The establishment of partial caches in different network locations favors the establishment of disjoint delivery paths, facilitating the distribution of forwarding load and the localization of traffic. However, these benefits come at the cost of forwarding state which increases with the size of the resulting forest.

At a second level, pieces are further partitioned into blocks, again as in BitTorrent. This second level of fragmentation facilitates the provision of data from multiple sources, i.e. as explained in the previous section, an OAR may join a multicast tree while data are in transit, in which case the remainder of the piece shall be provided by a cache. As shown in Figure 2, this further fragmentation enables the provision of the first four blocks only to OAR *3* from OAR *1*'s cache.

III. Evaluation

In order to provide a realistic application model for the evaluation of the proposed architecture, we have designed a MultiCache-based content distribution application that can be directly compared to regular BitTorrent. In this application a content provider employs content fragmentation to create multiple trees for the delivery of a single file. All identifiers are retrieved by end-hosts via out-of-band means, e.g., a MultiCache-torrent file. In order to reduce forwarding dependencies [9], piece identifiers are assumed to have been appropriately selected so that the RV functionality is provided by OARs residing at the content provider's domain. Upon arrival to the network, end-hosts connect to their proxy OAR and submit requests for pieces of the file. The number of pending requests is capped, in an analogy to regular BitTorrent. Each node submits its requests independently of other end-hosts, since we cannot assume any form of collaboration between end-hosts. Once a piece has been entirely downloaded, the next piece is requested from the proxy OAR until the file download has completed.

A. Simulation Environment

The evaluation of MultiCache is based on a detailed full stack simulation environment based on the OMNeT++ Simulator [10] and the OverSim Framework [11]. The MultiCache content distribution application is compared against our own BitTorrent implementation for OMNeT++[12]. In our simulations we used Internet-like topologies generated by the popular *Georgia Tech Internet Topology Model* (GT-ITM). For our measurements, we created topologies comprised of 1225 routers hierarchically organized in 25 stub and 5 transit domains. In all topologies, the default link establishment probabilities were used. All scenarios include the download of a single 256 MB file by 100 end-hosts attached to randomly chosen stub routers, following the exponential decreasing arrival rate process described in [13]. We used the default parameters for both the Peer-Wire and Tracker protocols of BitTorrent. In the case of MultiCache, we use 16KB blocks and set the default size of a piece to 16MB.

B. Results

The following results express the average of five repetitions of each simulation scenario with different random number generator seeds, over five different topology instances.

1) Traffic: Focusing first on the potential reduction of unnecessary transmissions, we investigate MultiCache's performance with respect to inter-domain and intra-domain traffic. In the former case we measure the total number of bytes of egress traffic at each stub domain, while in the latter we measure the total link stress at each stub domain, i.e. the aggregate number of block transmissions over all links of a domain. Due to length limitations, Figure 3 presents the cumulative distribution of both metrics over the set of stub domains in the considered topologies. The gains incurred by MultiCache functionality are in both cases substantial, reaching an average decrease of 57%-60% compared with BitTorrent. We attribute this reduction to the localization of traffic due to the deployment of caches. Indeed, the average distance traveled by a BitTorrent block was 8.86 hops, while MultiCache blocks only traversed 4.61 consecutive links before reaching an end host.

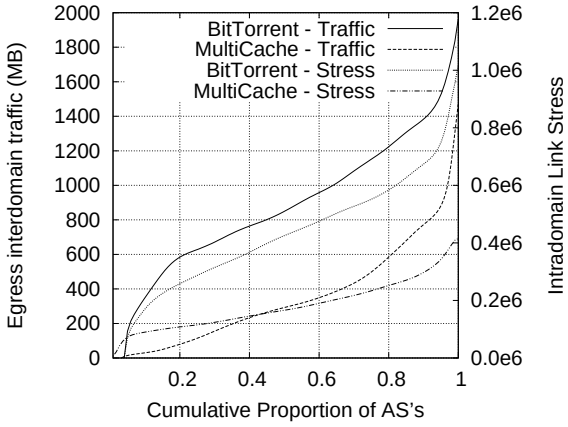

Fig. 3. Cumulative distribution of egress interdomain traffic and intradomain link stress

2) Download time: Even though the reduction of network traffic is of particular importance to network operators, QoS at the end users must not be neglected. Here we express QoS as the download time experienced by end users. Figure 4 illustrates the download times observed with BitTorrent and MultiCache. The download time perceived with MultiCache is on average 88% lower than in the case of BitTorrent. This huge reduction is due to three important factors. First, in the case of MultiCache end-hosts do not engage in a search for the required data among participating peers. This is a direct consequence of the information-centric model, in which users

simply request data from the network. Second, the download rate of end hosts is not capped by the uplink of their peers but by the forwarding capacity of the OARs, which is typically higher. Finally, our current assumption on infinite cache space results in the elimination of cache misses, yielding only an upper bound for MultiCache's performance.

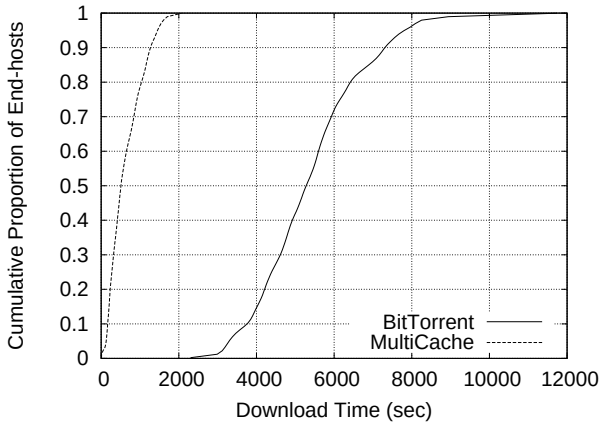

Fig. 4. Cumulative distribution of download time

IV. Related work

The most closely related approach to MultiCache is the LSAM Proxy Cache scheme, where multicast is used to distribute popular web pages to proxy caches [14]. A major differentiation is that in LSAM caches are only fed via multicast, while in MultiCache caches may be fed by other caches, resulting in traffic localization and further reduction of content provider load. In the same direction, MultiCache's anycast functionality enables the automated discovery of closely located cached objects, replacing, in a sense, the proactive push of content towards higher layers of the multicast tree. SplitStream [7] stripes content to produce a forest of disjoint Scribe trees in an effort to efficiently distribute the forwarding load. While MultiCache is orthogonal to this effort, we note that the employed tree-reconfiguration mechanisms may result in parent-child relationships violating the locality-related properties MultiCache is based on [15]. ChunkySpread[16] is also based on the creation of multiple trees but with a P2P orientation. Unlike both these approaches, MultiCache focuses on the localization of traffic and the reduction of network traffic both across and inside domain boundaries.

V. Conclusions and Future work

In this paper we presented MultiCache, an overlay information-centric architecture based on multicast and caching. Our preliminary results show that the proposed architecture has the potential to localize network traffic, relieving network operators from costly interdomain transmissions and further reducing traffic inside their domains. At the same time the localization of traffic and the parallelization of data flows acts in favor of end-hosts that perceive substantially reduced download times. However, the presented results only provide a baseline for the performance of the proposed architecture. Our current work focuses on the implementation and evaluation of appropriate cache replacement schemes.

Acknowledgment

The work reported in this paper was supported by the ICT PSIRP project under contract ICT-2007-216173.

References

[1] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet service providers fear peer-assisted content distribution?" in *Proc. of the ACM/USENIX Internet Measurement Conference*, 2005, pp. 63–76.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE JSAC*, vol. 20, no. 8, pp. 100–110, 2002.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. of the Middleware Conference*, 2001, pp. 329–350.

[4] R. Tewari, M. Dahlin, H. Vin, and J. Kay, "Design considerations for distributed caching on the internet," *Proc. of the ICDCS*, 1999.

[5] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented networks," in *Proc. of the ACM CoNEXT*, 2008, pp. 1–6.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scalable application-level anycast for highly dynamic groups," in *Proc. of the Networked Group Communication Workshop*, 2003.

[7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in cooperative environments," in *Proc. of the ACM SOSP*, 2003, pp. 298–313.

[8] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1447–1460, 2008.

[9] C. Diot, B. Levine, B. Lyles, H., Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *IEEE Network*, vol. 14, no. 1, pp. 78–88, 2000.

[10] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *Proc. of the ICST SIMUTools*, 2008, pp. 1–10.

[11] I. Baumgart, B. Heep, and S. Krause, "OverSim: A flexible overlay network simulation framework," in *Proc. of the IEEE Global Internet Symposium*, 2007, pp. 79–84.

[12] K. Konstantinos, V. Kemerlis, C. Stais, and G. Xylomenos, "A BitTorrent Module for the OMNeT++ Simulator," in *Proc. of the IEEE MASCOTS*, 2009, pp. 361–370.

[13] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "A performance study of BitTorrent-like peer-to-peer systems," *IEEE JSAC*, vol. 25, no. 1, pp. 155–169, 2007.

[14] J. Touch and A. Hughes, "LSAM proxy cache: a multicast distributed virtual cache," *Computer Networks and ISDN Systems*, vol. 30, no. 22-23, pp. 2245–2252, 1998.

[15] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang, "The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols," in *Proc. of the IPTPS*, 2005.

[16] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast," in *Proc. of the IEEE ICNP*, 2006, pp. 2–11.