A hybrid overlay multicast and caching scheme 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 long been realized that the use of the Internet has moved away from its original end-host centric model. The vast majority of services and applications is nowadays focused on information itself rather on the end-points providing/consuming it. However, the underlying network architecture still focuses on enabling the communication between pairs of end-hosts, leading to a series of problems, such as the inefficient utilization of network resources, demonstrated by the proliferation of peer-to-peer (P2P) and file sharing applications. In essence, the prevailing end-to-end nature of the current Internet architecture prohibits network operators from controlling the traffic carried by their networks, leaving this control entirely to end users and their applications. In this paper, we investigate the potential benefits of MultiCache, an overlay network architecture aiming at handing control back to network operators. In MultiCache proxy overlay routers enable the delivery of data either via direct multicast, or via multicast fed caches residing at the leaves of multicast delivery trees. We study crucial aspects of our architecture, paying special attention to the properties of our distributed caching scheme, and investigate the feasibility of a progressive deployment of the proposed functionality over the existing Internet.**

I. INTRODUCTION

The Internet emerged as a communications substrate enabling the delivery of data between pairs of end hosts. Though suitable for the once prevailing end-to-end communication patterns, this end point centric model seems to no longer cater to current communications needs. End users are now primarily concerned with accessing a desired piece of information rather than the specific end point providing it. The proliferation of CDN services, P2P applications, cloud computing applications, etc., demonstrates this radical shift towards information centrism. However, this shift has not been reflected in a corresponding adjustment of the underlying network model. This lack of information awareness inside the network means that end points are being organized into overlay content delivery structures that are inherently agnostic of the underlying network topology. This has led to a series of inefficiencies regarding the use of network resources and is posing significant challenges to network operators in controlling the traffic traversing their networks [1]. In the characteristic example of P2P applications that dominate Internet traffic, a multitude of redundant packet transmissions takes place due to network agnostic decisions taken at the edges [2].

In view of this situation, the need for a radical shift towards an *information-centric* networking model has become apparent, giving rise to several important research initiatives (e.g., [3], [4]). In the same vein, we have proposed MultiCache, an overlay network architecture aiming to bring information into focus [5]. In MultiCache network operators deploy overlay access routers using the Pastry overlay routing substrate [6]; these routers dynamically establish forests of Scribe overlay multicast trees [7] for the delivery of fragmented data objects to synchronously submitted requests (e.g. flash crowds). In addition, the delivered fragments are cached at the leaves of the corresponding multicast trees, resulting in the availability of each data fragment in multiple cache locations. The established multicast forwarding state is used to locate caching points in an anycast fashion, taking advantage of the locality awareness of the overlay routing substrate, ultimately leading to data being delivered either by a nearby cache location, or via overlay multicast if the content has not been cached.

A preliminary performance evaluation of MultiCache and comparison against BitTorrent has shown its potential in reducing network traffic while providing considerably better download times [5]. However, these results only served in gaining insight on the potential benefits and in defining an upper limit for the expected gains, as they assumed caches of infinite size. In this paper, we take a step further and focus on the properties of MultiCache's distributed caching scheme. We investigate the impact of limited cache sizes and study several cache replacement strategies. Moreover, we discuss the effect of deployment density on MultiCache performance, aiming to explore the relationship between traffic gains and required investment in equipment. Finally, we explore MultiCache's ability to localize traffic with respect to the distribution of content requests across the network.

The remainder of this paper is organized as follows. In Section II we provide a description of MultiCache, focusing on the proposed caching scheme. We then present our evaluation framework and discuss our simulation results in Section III. In Section IV we present previous research related to our work. Finally, we describe our next steps and conclude in Section V.

II. MULTICACHE ARCHITECTURE

MultiCache functionality is deployed in an overlay fashion inside access networks. In particular, this entails the deployment of additional infrastructure in the form of *overlay*

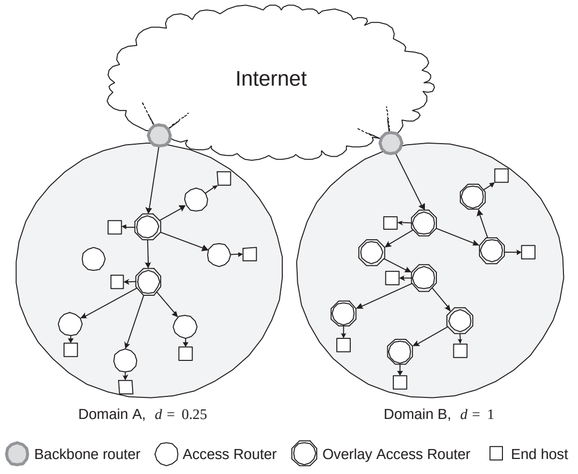


Fig. 1. MultiCache deployment

access routers (OARs), possibly collocated with regular access routers. All deployed OARs participate in the Pastry overlay routing and forwarding substrate. In addition, OARs act as proxies of end-hosts in the overlay, i.e., upon attachment to the network, 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. Its role is to act as the interface of the end-host to the overlay, possibly aggregating data requests from multiple attached end-hosts. A simple MultiCache deployment example is given in Figure 1.

A. Multicast

Multicast forwarding takes place among OARs driven by end-host requests, i.e., each end-host issues requests for desired data objects to its proxy OAR via their established connection. If the proxy OAR is not already part of the Scribe generated multicast distribution tree for that data, these requests are translated to Scribe JOIN messages so as to allow the proxy OAR to become part of the corresponding Scribe tree. The joining process deviates slightly from regular Scribe in that the JOIN messages are extended to further carry the IP address, listening port number, credentials of the initial issuer of the JOIN message (i.e., the proxy OAR[1]) and the 32-bit *Autonomous System* (AS) Number of the proxy OAR's AS [8]. 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 Scribe JOIN message eventually reaches the *Rendez Vous* (RV) point designated by Scribe for the group corresponding to the requested data, the content provider will be solicited to deliver the data via the created multicast tree. It is assumed that the content provider has already created the respective group, and therefore has contacted the RV point, before announcing data availability. Due to the asynchronous character of request arrivals, this process may result in partial data availability at the leaves of the multicast tree at the end of the multicast session. However, the caching mechanism ensures the completion of these partial feeds.

B. Caching

In MultiCache, caches are located at proxy OARs, i.e., at the leaves of multicast trees. The same content may be cached at multiple network locations close to the clients. This facilitates the discovery of caches in the networking vicinity of a requesting OAR and enables the localization of traffic (see Section II-B.4). It is also noted that placing caches at the edge of the network avoids incentive incompatibilities regarding inter-AS relationships as discussed in [9].

1) Cache discovery: In order to locate an available cache, MultiCache uses the already established overlay multicast forwarding state. The OARs cache the forwarding state established during tree creation even after the end of a multicast transmission. 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 downstream and allow their discovery upon cache requests. Note that caches may be fed by other caches, therefore OARs cannot rely on forwarded traffic in order to deduce cache availability below them. OARs further propagate received CACHE UPDATE messages towards the root *iff* they have not already done so for another downstream cache, thus avoiding feedback implosion.

When an end-host requests data, the Scribe JOIN message sent by its proxy OAR is suppressed at the first OAR that has already joined the respective tree, henceforth termed as a *meta-cache* OAR. What happens next, depends on the state of the meta-cache OAR with respect to the indicated object. If the requested object has been cached by the meta-cache OAR itself, the cached data will be directly delivered to the requesting node (direct cache hit). If the data are not cached but the meta-cache OAR has previously completed forwarding the data object to its descendants[2], it will anycast a CACHE REQUEST message to the sub-tree below it in a *depth first search* (DFS) fashion, carrying all extra information inserted in the JOIN message. In particular, at each level of the traversed sub-tree this 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 requesting proxy OAR. Among equivalent candidates, a randomized selection ensures the uniform distribution of load to the available caches. Eventually, a CACHE REQUEST reaches an OAR that has cached the data, and a TCP connection is established between the caching OAR and the proxy OAR for the delivery of the cached data.

Finally, if the meta-cache OAR is currently forwarding the requested data, it forwards the arriving multicast data to the joining node, keeping also track of the part of the data object that was not delivered due to the late arrival of the JOIN message. Upon the arrival of the first CACHE UPDATE

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

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

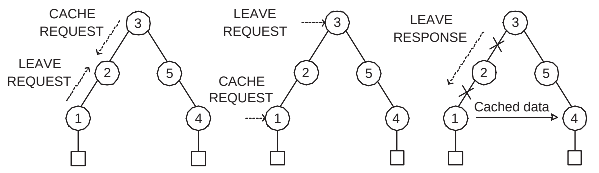


Fig. 2. MultiCache leave procedure

notification a data receiver, a CACHE REQUEST is issued for the missing data for each partially served child of a meta-cache OAR, thereby reverting to the previous case.

2) Cache eviction: In MultiCache, cache availability is correlated with the overlay multicast forwarding state. This allows requests for content to lead to either the multicast-based delivery of data or to a cache hit, while preserving the locality properties of the established tree structure (see Section II-B.4) and avoiding extra control overhead for the discovery of cached objects. In practice, this means that cached items are not evicted from a cache unless the corresponding multicast forwarding state is torn down.

To synchronize caching and forwarding state, we have slightly altered Scribe's leave procedure. When a caching OAR issues a Scribe LEAVE message for the tree serving the cached object marked for eviction, this message propagates up the tree until either the first node with additional children or the RV point is encountered. Then a LEAVE RESPONSE message is issued towards the leaving child, thus tearing down the forwarding state and removing the cached data. In contrast, in regular Scribe the forwarding state is removed immediately upon the reception of a LEAVE message. The new procedure ensures that eventually, all requests for data reach either a caching OAR (in the form of CACHE REQUEST messages) or the root if no other cache location is available (in the form of Scribe JOIN messages). In the latter case, the content provider is solicited to provide the desired object again via the established multicast delivery path. The above procedure is illustrated in the simple example of Figure 2. Node *2* will not remove node *1* from its forwarding table until a LEAVE RESPONSE message is received from node *3*. In the meanwhile, node *4*'s issued CACHE REQUEST will be normally served.

3) Cache replacement: As mentioned above, OARs always cache the content delivered due to an end host request. When a request arrives at an OAR with an exhausted cache space, a cache replacement policy is employed to select an item for eviction. Common replacement policies (e.g., *Least Recently Used* (LRU)) aim at adjusting cache contents to request patterns so that less popular items leave space for more popular ones, thus increasing the cache hit ratio. In MultiCache, cache replacement aims at taking advantage of the multiplicity of cache locations inside an administrative domain. To this end, caching OARs keep track of the popularity of each cached item with respect to the frequency and/or recency of hits from other OARs inside the domain. Since all content delivered to an OAR is locally cached, such hits imply that additional copies of the same object are probably cached nearby. Hence, in MultiCache we examine the suitability of the *Most Recently Used* (MRU) and *Most Frequently Used* (MFU) policies; these policies favor the selection of items that are most likely to be available at other cache locations. The rationale behind the MRU policy is that the most recently served object is more likely to be still available at the served OAR, i.e., not to have been evicted yet, therefore the existence of an alternative cache location allows replacing that item. In the case of MFU, the probability of eviction increases with the anticipated number of alternative cache locations.

It must be stressed that common replacement policies such as LRU and LFU refer to the recency/frequency of requests referring to the entire data item (file). In MultiCache, caching, and therefore cache replacement, takes place at a fragment level (see Section II-C), allowing the partial caching of files. The examined MFU and MRU policies aim at reflecting the existence of specific cache locations and therefore are enforced on fragments, regardless of the data item that each fragment belongs to. In this manner there is no need for control signaling and state overhead for the association of single pieces with the corresponding data items. We study the behavior of these policies in section III-C.

4) Locality properties: MultiCache favors localized cache hits by building upon Pastry's locality properties and the multiplicity of cache locations. According to Pastry's route convergence property, since caching OARs are essentially leaves of the data item's Scribe multicast tree, a Scribe JOIN message from a proxy OAR is expected to reach a meta-cache OAR at a distance approximately equal to the distance between the proxy OAR and 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 [10], in cases of multiple cache locations, Scribe JOIN messages from proxy OARs are expected to first reach nearby meta-cache OARs, thus leading to closely located 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 enhanced by the simple AS number-based cache selection mechanism.

C. Content fragmentation

MultiCache allows the fragmentation of large files into *pieces*, in a BitTorrent fashion, leading to the creation of a forest of Scribe trees, resembling SplitStream [11], but without the explicit goal of creating disjoint trees. This fragmentation serves several important goals. First, it facilitates the establishment of parallel data flows towards a recipient node, possibly exploiting the available downlink bandwidth and avoiding the sequential delivery of large files. Furthermore, it allows the partial caching of large data volumes, i.e., certain pieces can be cached independently of others, enabling the fine grained management of caching space [12]. 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. Pieces are further partitioned into blocks, again as in BitTorrent fashion. This second level of fragmentation facilitates the provision of data from multiple sources. For example, as explained in the previous section, an OAR may join a multicast tree while data are in transit, in which case the first part of the piece shall be later provided by a cache.

III. Performance Evaluation

In order to provide a realistic, information-centric application model for the evaluation of the proposed architecture, we designed a simple MultiCache-based content distribution application borrowing features from 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 [13], piece identifiers are assumed to have been appropriately selected so that the RV points will be 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, similarly to regular BitTorrent. Each node submits its requests independently: as in BitTorrent, we do not assume any form of collaboration between end-hosts.

A. Simulation Environment

The following evaluation is based on a detailed simulation environment developed over the OMNeT++ Simulator [14] and the OverSim Framework [15]. In our simulations we used Internet-like topologies generated by the *Georgia Tech Internet Topology Model* (GT-ITM) [16]. 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.

In order to study the properties of our caching scheme, we generated synthetic traces of request arrivals for several files across the network. To generate this workload we used features of the ProWGen trace generation tool [17]. To better reflect the characteristics of a P2P application we replaced the Zipf distribution of file popularities with the Mandelbrot-Zipf distribution proposed in [12]. A certain number of requests is generated for each file in the workload, according to the file's popularity. All file requests follow the exponentially decreasing arrival rate process described in [18], parameterized according to the popularity of the corresponding file. We interleave these single file traces by placing the first request of each file at a constant time interval after the first request for the previous file. This reflects the constant torrent arrival rate observed with BitTorrent in [18]. File sizes were sampled from the traces in [19]. The content providers, one per file, are uniformly distributed across the entire network. Each of the generated requests is then assigned to one of the 100 end hosts we attach at a randomly chosen access router of the topology.

B. Evaluation framework

In order to study the properties of MultiCache's caching scheme, our first metric is the achieved *cache hit ratio* (CHR). To further study whether traffic is localized within AS boundaries, we also measure the *intra-domain cache hit ratio* (CHR-Intra) which reflects only cache hits on OARs residing in the same AS as the end-host receiving the cached data. Finally, we measure the *distance to block source*, i.e., the average number of physical hops traversed by blocks arriving at end hosts, in order to assess the overall locality properties of data transfers.

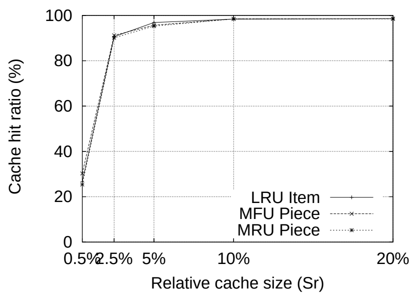
These metrics are studied for the cache replacement policies described in Section II-B.3, as well as for various cache sizes. Following the methodology in [20], we consider relative cache sizes (S_r), i.e., the cache size is expressed as a fraction of the "infinite cache size", which is the minimum cache size required to avoid replacements. We also examine the effect on these metrics of MultiCache deployment density $d \in [0, 1]$, defined as the fraction of the access routers that are enhanced with MultiCache functionality. In the example deployment of Figure 1, the density for domain A is $d_A = 2/8 = 0.25$ while for domain B it is $d_B = 1$. In this paper, we assume uniform density values across all ASs.

Finally, we investigate the ability of MultiCache to localize traffic inside domain boundaries depending on the popularity of data objects inside the domain. For this reason we define the *localizability* parameter $l \in [0, 1]$ as an indicator of the concentration of end-hosts (and the corresponding requests) inside domain boundaries. If we denote as D the total number of administrative domains in the topology, then end hosts are uniformly distributed across $\max[(1 - l)D, 1]$ administrative domains. At $l = 1$, all end-hosts reside in the same AS, while at $l = 0$, they are uniformly distributed across all ASs.

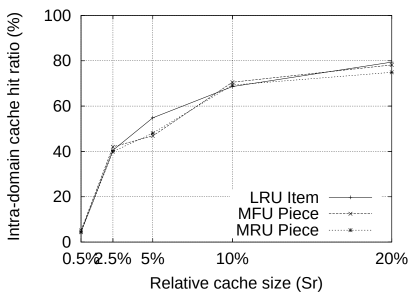
C. Results

1) Cache size and replacement policies: Figures 3(a) and 3(b) show the CHR and CHR-Intra achieved with the LRU, MFU and MRU cache replacement policies, for relative cache sizes (S_r from 0.5% to 20%). Interestingly, all these policies exhibit approximately the same behavior for all cache sizes considered. Note that the CHR reaches values up to 98.5% for higher S_r values, thus reducing the amount of data delivered via overlay multicast and taking advantage of available caches throughout the entire network. As a result, MultiCache reduces the impact of overlay multicast stretch and takes advantage of Pastry's proximity properties in order to locate nearby copies of the desired data. This is clearly demonstrated in Figure 3(c) which depicts the distance to block source. As the cache size increases, average distance decreases, denoting the delivery of content from nearby caches. Again, no cache replacement policy exhibits superior performance, leading us to the adoption of the MFU policy due to its implementation simplicity.

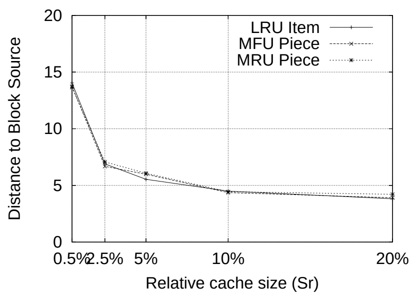
2) Deployment Density: Since MultiCache necessitates the deployment of additional infrastructure (OARs) by network operators, a crucial issue for the viability of the proposed architecture is the magnitude of the investment required. Figure III-C shows the cache hit ratio for various deployment densities and relative cache sizes. Figure 4(a) shows that even though CHR increases with deployment density, for relative cache sizes ranging from 2.5% to 20% of the "infinite cache



(a) Cache hit ratio

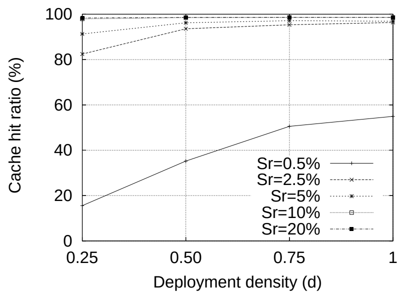


(b) Intradomain Cache hit ratio

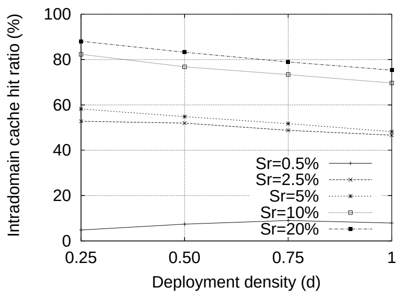


(c) Localization of traffic

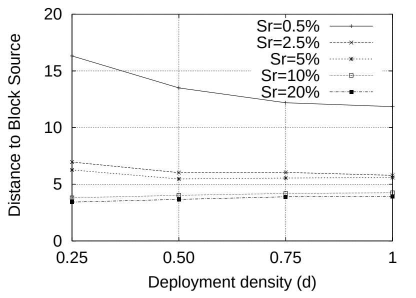
Fig. 3. Effect of cache replacement policies on cache hit ratio and the localization of traffic ($l = 0, d = 0.25$).



(a) Cache hit ratio



(b) Intradomain Cache hit ratio



(c) Localization of traffic

Fig. 4. Effect of deployment density on cache hit ratio and the localization of traffic ($l = 0.5, MFU$).

size" the perceived CHR is always greater than 80% at a deployment density of 25%. Figure 4(b) shows that CHR-Intra ranges from 46% to 88% for higher relative cache sizes, meaning that this portion of traffic is held inside domain boundaries. As the density increases however, local cache hits decrease, due to the reduced degree of request aggregation at caching OARs and the corresponding reduction of direct cache hits at proxy OARs. This is also depicted by the modest reduction of the average network distance travelled by data blocks, shown in Figure 4(c). While denser deployments result in more cache locations, and therefore shorter distances between proxy and caching OARs, they also mean that similar requests and the resulting cached content are distributed across a correspondingly larger number of locations. Therefore, the modest investment required to achieve a deployment density of 25 to 50% is sufficient to reap all the benefits of MultiCache.

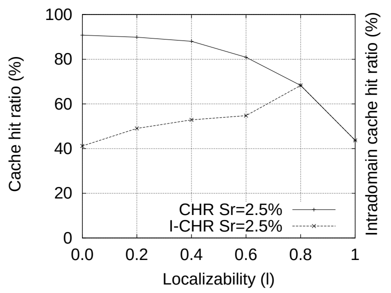


Fig. 5. Effect of localizability on cache hit ratio ($d = 0.25, S_r = 2.5\%, MFU$)

3) Localizability: Figure 5 shows the effect of the localizability factor on the cache hit ratios for S_r =2.5%[3]. Since the delivery of content to an OAR results in the availability of that content at the corresponding cache, localizability essentially expresses the availability of content inside domain boundaries. However, as localizability increases, so does the competition for caching space: the more data being delivered to a domain, the more content has to be cached locally within a fixed cache size. Hence, the content arrives at a domain but gets evicted due to the increased load on the caches. At the same time, content concentrates at intra-domain cache locations, due to co-located requests, therefore the portion of intra-domain cache hits rises, up to the point where all cached content is provisioned by a local cache. This means that MultiCache does takes advantage of localized request patterns, but only up to the point where cache size limitations do not allow further improvements.

IV. Related work

Povey *et al.* [21] pointed out some of the inherent inefficiencies of hierarchical caching and introduced the idea of employing only leaves of server hierarchies as caching locations, discovered through their ancestors. MultiCache departs from this approach in that it further employs multicast forwarding and builds on Pastry's proximity properties in order to localize traffic. In LSAM Proxy Cache, multicast is used to distribute popular web pages to proxy caches [22]. A major differentiation point of MultiCache is that caches are not only

[3]Similar results were derived for other relative cache sizes; they are omitted due to lengh limitations.

fed via multicast from the source, but also by other caches, resulting in the localization of traffic and the further reduction of the content provider load. Moreover, 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 [11] 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 tree-reconfiguration mechanisms employed in SplitStream may result in parent-child relationships violating the locality properties that MultiCache is based upon [23]. ChunkySpread[24] is also based on the creation of multiple trees but with a P2P orientation. Unlike both these approaches, MultiCache is deployed by network operators and 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 advance our work on MultiCache, an overlay information-centric architecture, focusing on its distributed caching scheme. Our findings show that MultiCache takes advantage of the multiplicity of cache locations, avoiding as far as possible the employment of overlay multicast for already transmitted content. Moreover, our results show that sparse MultiCache deployments can yield high intradomain cache hit ratios, thus localizing traffic inside domain boundaries.

Our current work focuses on interdomain cache service provision and specifically on the support of peering relationships between AS's. Our goal is to allow operators to share the benefits of their individual MultiCache deployments, while keeping control of the interdomain traffic generated by this form of collaboration. To this end, we aim at incorporating these peering relationships into Pastry, so that multicast forwarding and cache selection functions will implicitly take into account the willingness of network operators to exchange cache contents.

Acknowledgment

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

References

[1] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P users cooperate for improved performance?" *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 29–40, 2007.

[2] T. Karagiannis, P. Rodriguez, and K. Papagiannaki, "Should Internet service providers fear peer-assisted content distribution?" in *Proc. of ACM/USENIX IMC*, Berkeley, CA, USA, Oct 2005, pp. 63–76.

[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of ACM CoNEXT*, Dec. 2009.

[4] PSIRP Project, *PSIRP Project Home Page*, http://www.psirp.org.

[5] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "MultiCache: an incrementally deployable overlay architecture for information-centric networking," in *INFOCOM Work-in-Progress (WiP)*, San Diego, CA, USA, March 2010.

[6] 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.

[7] 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.

[8] IANA. (2009, Jun) Autonomous system (AS) numbers. [Online]. Available: http://www.iana.org/assignments/as-numbers/as-numbers.xml

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

[10] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scalable application-level anycast for highly dynamic groups," in *Proc. of NGC*, Sept. 2003.

[11] 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.

[12] 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.

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

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

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

[16] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. of the IEEE INFOCOM*, vol. 2, CA, USA, Mar 1996, pp. 594–602.

[17] M. Busari and C. Williamson, "ProWGen: a synthetic workload generation tool for simulation evaluation of web proxy caches," *Computer Networks*, vol. 38, no. 6, pp. 779–794, 2002.

[18] 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.

[19] A. Bellissimo, B. N. Levine, and P. Shenoy, "Exploring the use of bittorrent as the basis for a large trace repository," University of Massachusetts Amherst, Tech. Rep., June 2004.

[20] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary Cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[21] D. P. John and J. Harrison, "A distributed internet cache," in *In Proceedings of the 20th Australian Computer Science Conference*, 1997, pp. 5–7.

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

[23] 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 IPTPS*, February 2005.

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