# MultiCache: An overlay architecture for information-centric networking

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

***Abstract*—It has become apparent for quite some time that the Internet has evolved 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 and content delivery applications, it has not been reflected in a corresponding shift in network architecture. To address this issue, we designed MultiCache, an information-centric architecture aiming at the efficient use of network resources. MultiCache is based on two primitives: multicast and caching. It exploits overlay multicast as a means for content delivery and takes advantage of multicast forwarding information to locate, in an anycast fashion, nearby caches that have been themselves fed via multicast. We evaluate MultiCache against a widespread file sharing application (BitTorrent) with respect to both network resource consumption and end-user experience.**

***Index Terms*—Information centric, content centric, publish-subscribe, peer to peer, multicast, caching**

## I. Introduction

The Internet was originally designed as a communication substrate enabling the delivery of data between pairs of end-hosts. Unfortunately, this end-point centric model seems to no longer cater to current communications needs: while users focus on the desired information, the underlying network focuses on the end-to-end communication between end-hosts. Inevitably, a translation between the information domain and the networking domain must take place, typically consisting of the establishment of a delivery path between the data provider and the data consumer. This translation is usually inefficient, as it is based on end-point centric data delivery overlays that neglect network topology, data location and data popularity, ultimately over-consuming network resources. For example, in *Peer to Peer* (P2P) file sharing, it has been shown that a major part of the incurred traffic crosses *Internet Service Provider* (ISP) boundaries, even though the desired 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 via caching and multicast, are unlikely to succeed, as decisions are made at network end-points based on coarse grained information. We believe that only an information-centric network model can enable the efficient use of network resources, thus better reflecting user needs. In this context, by knowing *what* is being delivered, in addition to *where* it originates from or is destined to, the network becomes inherently capable of forming targeted and efficient delivery structures. In addition, in this model users can directly express their interest in pieces of information, rather than engaging in the aforementioned domain translation.

Towards this direction, we present *MultiCache*, an overlay network architecture that brings information into focus. MultiCache takes advantage of information-awareness to improve network utilization via resource sharing. To achieve this, 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 this 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 caches. By taking advantage of the locality awareness of the 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 via unicast.

In this paper, we highlight the benefits of information centric networking, as realized in MultiCache. In particular, we demonstrate how the proposed architecture takes advantage of information-awareness in order to improve both the utilization of network resources and the end-user experience. To this end, we compare a MultiCache-based content distribution application with the popular BitTorrent application. Our simulation results demonstrate a significant reduction of traffic load, in conjunction with considerably lower download times. To the best of our knowledge, this is the first head to head comparison of an information centric architecture against the current Internet model in the context of content distribution, with respect to network resource utilization and end user experience.

The remainder of this paper is organized as follows: In Section II we provide a thorough description of MultiCache, including both protocol functionality and deployment issues. We then present our MultiCache-based content distribution application, comparing it with BitTorrent in Section III. In

Section IV, we analytically investigate the scaling properties of MultiCache with respect to the overlay multicast trees created and the forwarding state required by MultiCache. We contrast MultiCache with other approaches in Section V, concluding and presenting our next steps in Section VI.

## II. Proposed Architecture

### A. Overview

MultiCache aims to establish an information-centric model of communication that better reflects current Internet usage patterns, in order to facilitate the deployment of resource sharing mechanisms. The introduction of information-awareness into the network enables the network (in addition to the end points) to identify pieces of information. In consequence, request similarities can be detected, thus allowing request aggregation and, eventually, network resource sharing. This enables the deployment of resource sharing mechanisms inside the network, rather than at the end-points. At the same time, information-awareness simplifies the usage model, as it allows end-hosts to directly denote the desired piece of information to the network, rather than engaging in the process of locating an end-point providing it.

In this context, the proposed architecture is realized as an overlay network, based on the deployment of additional infrastructure inside access networks. Namely, ISPs deploy and control *Overlay Access Routers* (OARs) at their *Points-of-Presence* (PoPs). These OARs establish an overlay routing substrate for the forwarding of data based on information identifiers, using the Pastry routing scheme [3]. Pastry provides a location-aware key-based routing fabric (see Section II-B1) which allows the network location of any piece of information to be discovered, given a globally unique information identifier (ID) in the *Pastry* namespace. Based on this substrate, the proposed architecture adopts and extends the Scribe scheme [2] to enable the joint operation of overlay multicast and caching, subject to the temporal characteristics of the requests (see Sections II-B2, II-C and II-D). Scribe establishes multicast trees to serve multiple synchronous requests, while feeding in-network caches also located at the OARs. The already established multicast forwarding state is reused to allow later requests to reach nearby caches, leading to a hybrid protocol where data is delivered either via multicast or from a caching location via unicast.

At the edges of the network, end-hosts access the overlay network through a *proxy* OAR, designated during network attachment. The usage model of the proposed architecture closely follows the publish/subscribe paradigm [4]. Data consumers (i.e., subscribers) send a subscription request message towards their proxy OAR declaring the ID of a desired piece of information. The proxy OAR is then responsible to fetch the requested item using multicast and/or caching. On the other hand, data providers (i.e., publishers) advertise their content to the network by submitting an advertisement message to their proxy OAR, which is then responsible to locate the corresponding *Scribe* tree, using *Pastry* routing (see Section II-B). Despite not participating in any of the overlay protocols (i.e., Pastry, Scribe and MultiCache protocols), end-hosts interact with the network in a simplified manner that does not include a translation between the desired data and its location.

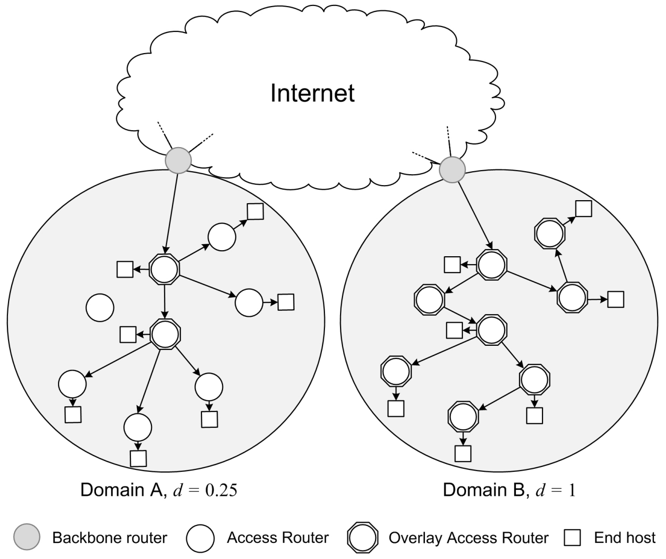


Fig. 1. MultiCache Deployment.

The proposed deployment of overlay functionality inside access networks is motivated by several factors. First, it is expected to significantly improve performance, as it results in multicast trees that avoid forwarding content over the, typically lower bandwidth, access uplinks [5]. Moreover, the overlay character of the architecture facilitates deployment, as it does not require the replacement of existing infrastructure, thus also allowing the unobstructed operation of established services and applications. By deploying MultiCache inside access networks, content is cached close to the clients, facilitating the discovery of caches in the clients' networking vicinity and therefore enabling the localization of traffic (see Sections II-D3 and III). At the same time, the deployment on top of ISP owned, dedicated servers is expected to provide lower churn rates, mitigating the maintenance overhead of the routing substrate, in contrast with typical end host deployment scenarios [6]. Finally, as discussed in [7], 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 collocated with the corresponding access routers.

### B. Background

We provide below an introduction to the *Distributed Hash Table* (DHT) based Pastry overlay routing substrate [3] and to the Scribe overlay multicast scheme [2] upon which the MultiCache functionality is built.

*1) Pastry:* In Pastry, as in all DHT based routing substrates, a flat identifier space is uniformly distributed among nodes. Pastry routes a messages destined to a specific identifier to the node responsible for that identifier. Each node maintains routing state that enables message forwarding using prefix based routing: at each routing step, a node forwards the message to another node whose identifier shares at least one more digit with the target identifier. Pastry takes network locality into consideration during routing state computations, i.e., among equally qualified routing table entries, the one corresponding to the closest node with respect to the employed proximity metric, e.g., hop count or RTT, is selected. By reducing overlay path stretch, this yields Pastry's *short routes* property and constitutes one of the reasons for selecting Pastry as the routing substrate for MultiCache. Additionally, according to Pastry's *route convergence* property, the distance traveled by two messages originating from two distinct nodes before their routes converge towards the same destination, tends to be approximately equal to the distance between the two source nodes in the proximity space. As described below in detail, our caching scheme takes advantage of this property in order to create caches residing in the network vicinity of end-hosts.

*2) Scribe:* Scribe enables multicast distribution by mapping the name of each group to an identifier and making the node responsible for that identifier the *rendezvous* (RV) point of the group. Receivers join the group by sending a join message towards the group identifier; as the message propagates towards the RV point, reverse path routing state is established until a node already in the tree is found, thus forming a multicast tree rooted at the RV point. A sender simply routes data towards the group identifier, so that the RV point may then propagate it over the established tree. An important characteristic of Scribe, motivating its adoption for MultiCache, is that multicast routing state is decentralized: each node in a tree is only aware of its immediate ancestors and descendants. This presents a significant scalability advantage over other schemes (e.g., Bayeux [8]) as it means that Scribe does not require excessive signaling in order to gather global state information. Moreover, it must be noted that Scribe follows the publish/subscribe paradigm [4] which is considered especially suitable for information-centric networking, in that it decouples the sender from the receiver [9].

### C. Multicast

In MultiCache, multicast forwarding takes place among OARs driven by end-host requests, i.e., end-hosts issue requests for data objects to their proxy OARs via control connections. These requests may then be translated to corresponding Scribe JOIN messages, depending on what the proxy OAR knows about the data item requested. The joining process deviates slightly from regular Scribe in that JOIN messages are extended to further carry the IP address, the listening port, the credentials and the 32-bit *Autonomous System* (AS) number [10] of the initial issuer of the JOIN message (i.e., the proxy OAR of the end-host). Note that in the case of multiple overlay hop paths, this proxy OAR may not be the OAR that eventually delivers the JOIN message to an already joined node. This information is used during cache searching and provisioning, as explained below.

During the joining process, the OARs establish TCP connections with their descendants in the tree, to ensure the reliable delivery of the requested data. When a JOIN message eventually reaches the RV point, the content provider is solicited to start delivering data, which then starts traversing the tree formed by the TCP connections. Note that we assume that the content provider has already created the respective group, and has therefore established contact with the RV point. Due to the asynchronous character of request arrivals, this process may result in partial data availability at the leafs of the tree at the end of a 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 subfigures depict progressive snapshots of a simple Scribe tree. The arrays below each OAR denote the availability of content at each OAR and will be further explained in Section II-E. OAR *1* first joins a Scribe multicast tree via OAR *2*, followed by OARs *3* and then *5* which join the tree via OAR *4* during the multicast session. At the end of the multicast session (end of Step III), OARs *3* and *5* have received only part of the multicasted content, i.e., they are missing blocks 0-3 and 0-6 respectively. OAR *6* joins after the end of the multicast session, thus receiving no blocks from it.

### D. Caching

Caching plays a vital role in MultiCache. Our goal is to take advantage of the constantly decreasing cost of storage space [11] in order to reduce network traffic via localized cache hits. In MultiCache the proxy OARs, located at the leaves of the multicast trees, cache the content they receive via multicast or unicast from other caches, leading to a distribution of caching locations across the network. We describe below our caching scheme in terms of cache discovery, indexing state management and cache space management.

*1) Cache discovery:* MultiCache uses the already established overlay multicast forwarding state to locate caches. The OARs maintain 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 point of the multicast tree. The purpose of these messages is to notify tree ancestors about the availability of cached items downstream, thus allowing their discovery upon cache requests. Caches may be fed by other caches, therefore OARs cannot solely rely on forwarded traffic to deduce cache availability in their descendants. OARs propagate received CACHE UPDATE messages towards the RV point *iff* they have not already done so for another downstream cache, thus avoiding feedback implosion.

When an end-host requests a data object, its proxy OAR creates a Scribe JOIN message, unless it already has a cached copy (direct cache hit). As in regular Scribe, this JOIN message 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 object has been cached by the meta-cache OAR itself, the cached data will be directly delivered to the requesting node. If the object has not been cached but the meta-cache OAR has previously forwarded it to its descendants, it will anycast a CACHE REQUEST message to the sub-tree below it in a *depth first search* (DFS) fashion, carrying the information inserted in the JOIN message. At each level of the traversed sub-tree, this

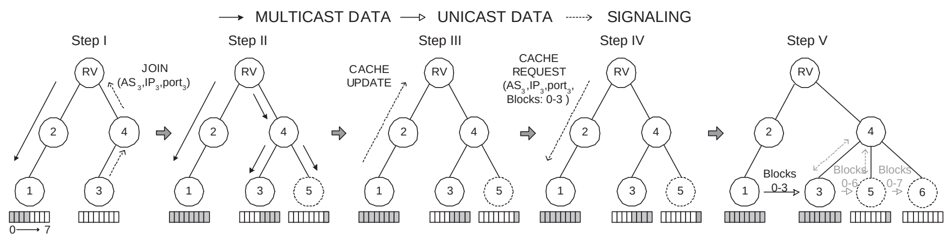


Fig. 2. MultiCache example.

message is forwarded to one of the children that have previously issued a CACHE UPDATE message, giving precedence to children belonging to the same AS as the requesting proxy OAR. Among equivalent candidates, randomized selection ensures a uniform distribution of load. Eventually, the CACHE REQUEST reaches an OAR that has cached the data, and a TCP connection is established between the caching OAR and the proxy OAR to deliver the cached data.

If, however, the meta-cache OAR is *currently* forwarding the requested data when it receives a JOIN, it starts forwarding the arriving multicast data to the joining node, keeping 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 notification from a data receiver, the meta-cache OAR issues a CACHE REQUEST for the missing data for each of its partially served children, thereby reverting to the previous case.

In the example of Figure 2, upon 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* notifies its parent with a CACHE UPDATE message which is eventually suppressed by the RV point (Step III). At this point, having partially served its descendants, the RV point 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 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*, which decides to send a CACHE REQUEST message to OAR *5*, thus resulting in localized traffic between OARs *5* and *6*.

*2) Cache eviction:* In MultiCache, cache availability is correlated with the overlay multicast forwarding state. This allows requests for content to lead to either multicast-based delivery or to cache hits, while preserving the locality properties of the established tree structure (see Section II-D3) and obviating the need for extra control overhead to discover cached objects. In practice, this means that cached items are not evicted from a cache, unless the corresponding multicast forwarding state is also torn down. Cache eviction is triggered either due to a need for cache replacement, or due to the invalidation of the cached content. In the latter case, we consider a *time-to-live* (TTL) mechanism to trigger cache eviction upon expiration. The estimation of the appropriate TTL value is application specific and out of the scope of this paper. In our content distribution application described in Section III, we consider non-versioned content (e.g., media files) with infinite TTL values.

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 a cached object marked for eviction, this message propagates upstream until either the first node with additional children or the RV point is encountered. A LEAVE RESPONSE message is then sent to the leaving OAR, 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. This procedure ensures that all requests for data reach either a caching OAR (CACHE REQUEST messages) or the RV point (Scribe JOIN messages) if no other cache location is available. In the latter case, the content provider is solicited to provide the desired object again via the established multicast delivery path.

*3) Cache replacement:* As mentioned above, OARs always cache the content delivered due to end-host requests. When cache space is exhausted, a new request triggers a cache replacement policy to select an item for eviction. Common replacement policies, such as *Least Recently Used* (LRU), attempt to adjust cache contents to request patterns, so that less popular items will make space for more popular ones, thus increasing the cache hit ratio. In MultiCache, cache replacement takes advantage of the multiplicity of cache locations inside an AS. Caching OARs track the frequency and/or recency of hits for each cached item from other OARs inside the same AS. Since all content delivered to an OAR is locally cached, these hits imply that additional copies of the same object are probably cached nearby. As a result, we also examine the suitability of the *Most Recently Used* (MRU) and *Most Frequently Used* (MFU) policies: these policies evict items that are most likely to be available at other cache locations.

It must be stressed that common replacement policies, such as LRU, refer to the recency/frequency of hits to an entire data item (file) (henceforth named as LRU Item). In MultiCache, caching, and therefore cache replacement, takes place at a fragment level (see Section II-E), allowing partial file caching. The MFU and MRU policies reflect the existence of additional cache locations and are therefore enforced on fragments, regardless of the data item that each fragment belongs to (henceforth named as MFU/MRU Piece). In this manner there is no need for control signaling and state management to associate fragments with the corresponding data items. We study the behavior of these policies in Section III-C3.

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 a 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. Furthermore, due to Pastry's prefix based routing, Scribe JOIN messages are initially expected to travel short distances at each overlay routing step. Hence, as demonstrated in [12], in cases of multiple cache locations, Scribe JOIN messages are expected to first reach nearby meta-cache OARs, thus leading to closely located caches. In effect, cache searches and cached data are expected to traverse short distances with respect to Pastry's proximity metric, leading to localized traffic. This is further enhanced by the simple AS number-based cache selection mechanism (see Section II-D1).

### E. Content fragmentation

MultiCache allows the fragmentation of large files into *pieces*, as in BitTorrent. This leads to the creation of a forest of Scribe trees, as in SplitStream [13] but without the explicit goal of creating disjoint trees. Fragmentation serves several important goals. First, it facilitates the establishment of parallel data flows towards a recipient node, potentially better exploiting the available downlink bandwidth. Second, it allows the partial caching of large data items, i.e., each piece can be cached independently of others, thus enabling fine grained management of the caching space [14]. Third, the establishment of partial caches at different network locations facilitates 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. 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 earlier (missing) blocks will be later provided by a cache.

## III. EVALUATION

As a realistic case study for the evaluation of our architecture, we designed a MultiCache-based content distribution application that can be directly compared with BitTorrent. In this application, a content provider employs content fragmentation to create multiple trees for the delivery of a single file. Fragment identifiers are retrieved by end-hosts out-of-band, e.g., a MultiCache-torrent file. To reduce forwarding dependencies [15], we assume that piece identifiers are selected so as to ensure that the RV point for each piece will be an OAR residing at the content provider's AS. End-hosts connect to their proxy OAR and submit requests for pieces of the file. The number of simultaneous pending requests is capped as in BitTorrent. Each node submits its requests independently of other end-hosts, since we cannot assume any form of collaboration between them.

Content distribution via BitTorrent is an obvious case of information-centric communication, forced by the current, information agnostic, Internet to be controlled by end-hosts. Our goals are, first, to examine the degree to which information awareness in the network, as introduced by MultiCache, enables a more efficient utilization of network resources and, second, to investigate how end user experience is affected. On the side, we also investigate the performance of MultiCache with respect to some key parameters.

### A. Simulation Environment

The evaluation of MultiCache is based on a detailed full stack simulation environment based on OMNeT++ [16] and OverSim [17]. The MultiCache content distribution application is compared against our own BitTorrent implementation for OMNeT++[18]. In both cases, the same setup was employed, including the network topology and link characteristics, the distribution of hosts across the network and the imposed workload. In our simulations we used Internet-like topologies generated by the *Georgia Tech Internet Topology Model* (GT-ITM). The simulated topologies comprised 1225 routers, hierarchically organized in 25 stub and 5 transit domains. In all cases, the default GT-ITM link establishment probabilities were used.

We generated synthetic request arrival traces for several files, using features of the ProWGen trace generation tool [19]. To better reflect the characteristics of our applications, we replaced the Zipf distribution of file popularities with the Mandelbrot-Zipf distribution, as proposed in [14]. A certain number of requests is generated for each file in the workload, depending on the file's popularity. The requests for each file follow the exponentially decreasing arrival rate process described in [20], parameterized according to the popularity of the corresponding file. We interleave in time 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, reflecting the constant torrent arrival rate observed in BitTorrent in [20]. File sizes were sampled from the traces in [21]. Content providers, one per file, are uniformly distributed across the entire network, and each generated request is then assigned to one of the 100 end-hosts we attached at randomly chosen access routers. We used the same workload for both MultiCache and BitTorrent.

We considered different bandwidth allocations for the uplink and downlink directions of access links, as current access technologies, such as ADSL, present this asymmetry. For the measurements presented below, we employed downlink bandwidth values ranging from 4 to 24 Mbps, and uplink bandwidth values of 1 or 2 Mbps. These values were distributed across end hosts as in [18]. Backbone routers were connected with 10 GBps links, while 1 GBps links connected access routers.

### B. Evaluation framework

The utilization of network resources is measured in terms of the *egress inter-domain traffic* (EIT) and *intra-domain traffic load* (ITL) incurred for the delivery of the content. The EIT metric reflects the total number of bytes of egress traffic at each stub AS, thus expressing the amount of traffic that network operators must pay for in order to reach a transit domain. The ITL metric measures the aggregate number of block transmissions over all links of an AS, allowing us to assess the load imposed within each administrative domain. The user experience is assessed by comparing the *download time* (DT) experienced by end-hosts in each case.

In order to explore the properties of our caching scheme, our first metric is the achieved *cache hit ratio* (CHR). To further study the localization of traffic within AS boundaries, we also measured the *intra-domain cache hit ratio* (CHR-Intra), which only reflects cache hits on OARs residing in the same AS as the requesting end-host. Finally, to quantify the locality of data transfers, we measured the *distance to block source* (DBS), that is, the average number of physical hops traversed by blocks arriving at end-hosts.

Following the methodology in [22], we considered relative cache sizes ($S_r$), i.e., cache size is expressed as a fraction of the "infinite cache size", or the minimum cache size required to avoid replacements. We also examined the effect on these metrics of the MultiCache *deployment density* parameter $d \in [0, 1]$, defined as the fraction of access routers 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 AS's. Unless otherwise stated, results refer to scenarios with $d = 0.5$, i.e., intermediate deployment density.

### C. Results

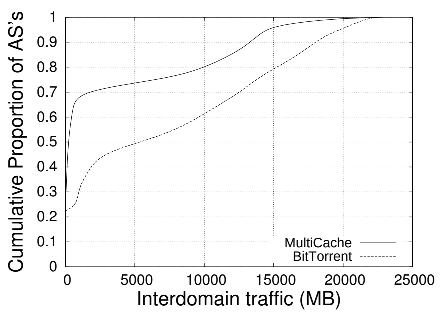
*1) Traffic:* Fig. 3(a) and 3(b) presents the CDF of the EIT and ITL metrics for the intermediate scenario of $d = 0.5$. In both cases, MultiCache achieves a substantial decrease in the traffic incurred: on average, it reduces EIT by 53.19% and ITL by 61.39%. This reduction is due to the localization of traffic through caching, as further demonstrated by the CHR metrics (see Section III-C4). At the same time, MultiCache avoids the exchange of data between end-hosts as in BitTorrent. This is evident in Figure 3(a), where we notice consistently low EIT values for almost 70% of the MultiCache enabled AS's, i.e., end-hosts do not generate inter-domain traffic in order to retrieve the desired content, instead taking advantage of previous downloads. This is a direct benefit of the information-centric network model and the caching mechanism employed, which enable network operators to regain control of the traffic carried by their networks. An emerging question then, is whether this incurs a penalty in the quality of service experienced by end users.

*2) User Experience:* Fig. 3(c) presents the CDF of the download times achieved with MultiCache and BitTorrent, over all downloaded items. The DT perceived with MultiCache is on average 62.64% lower than that of BitTorrent. This huge reduction is due to three factors. First, caching allows content to be locally stored and provided, as verified by the achieved CHR (see Section III-C4) and the average number of hops traversed by data blocks: with BitTorrent it is 8.6 hops, while with MultiCache it drops to approximately 6 hops. Second, with 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, where users simply request data from the network. Third, 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.

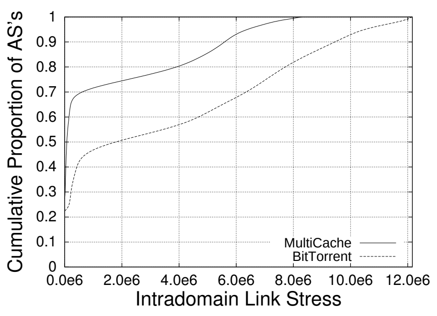
*3) Cache size and replacement policies:* Fig. 4(a) and 4(b) shows the CHR and CHR-Intra achieved with the LRU Item/Piece and MFU/MRU Piece cache replacement policies, for relative cache sizes ($S_r$) between 0.5% and 20%. Interestingly, all policies, except for LRU Piece, exhibit approximately the same behavior. As noted earlier, the LRU Item policy works at the item (file) level, capturing the popularity of the entire delivered item, while the MFU/MRU Piece policies work at the fragment level, reflecting the existence of additional caches throughout the domain. The LRU Piece policy however, despite operating at the fragment level, does not take advantage of the existence of additional cached replicas, resulting in a considerably lower CHR-Intra than the others. This is because it evicts items that are not widely cached in the domain, causing subsequent CACHE REQUEST messages to leave the domain. This is the reason for the increased DBS of the LRU Piece policy, as shown in Fig. 4(c).

In all cases, the CHR reaches values of up to 98.5% for higher $S_r$ values, thus reducing the amount of data delivered via overlay multicast by taking advantage of caches throughout the entire network. As a result, MultiCache reduces the impact of overlay multicast stretch, taking advantage of Pastry's proximity properties to locate nearby copies of the desired data. This is clearly demonstrated in Fig. 4(c): as the cache size increases, the average DBS decreases, as content is delivered from nearby caches. As all cache replacement policies perform similarly, we ended up adopting the MFU policy due to its simple implementation.

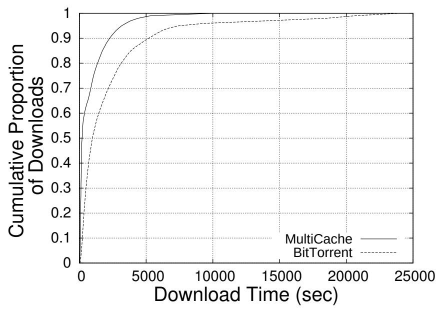
*4) Deployment Density:* Since MultiCache necessitates the deployment of additional infrastructure by network operators, in the form of OARs, a crucial issue for its viability is the required level of investment. Fig. 5(a) shows that even though the CHR increases with deployment density, for relative cache sizes ranging from 2.5% to 20% the perceived CHR is always greater than 80% at a deployment density of 25%; it does not drop below 60% even for the lowest deployment densities. Fig. 5(b) shows that CHR-Intra ranges from 35% to 88% for higher relative cache sizes, meaning that this portion of traffic does not leave the originating domain, thus explaining the observed reduction of EIT and ITL (see Section III-C1). This figure also reveals a tradeoff regarding the deployment of additional OARs. On the one hand, increased deployment density results in an increase of the total caching space available. On the other hand, dense deployments do not fascilitate request aggregation at proxy OARs, as they distribute the load among the proxy OARs, allowing CACHE REQUESTS to also leave the domain. Hence, the CHR-Intra initially increases with deployment density, but at a deployment density close to 18% it starts dropping again, due to the reduction of direct cache hits at proxy OARs. Therefore, the modest investment required to achieve a deployment density of 20-50% is sufficient to reap



(a) Egress inter-domain traffic (EIT)



(b) Intra-domain traffic load (ITL)



(c) Download time (DT)

Fig. 3. MultiCache content distribution application vs. BitTorrent ($d = 0.5$).

all the benefits of MultiCache.

## IV. Analysis

Introducing information-awareness into the network relies on the scalable representation and handling of information. In MultiCache, information is represented by flat identifiers handled in two layers. First, the underlying Pastry DHT is responsible for the key-based routing that enables the overlay multicast functionality. As demonstrated in [3], Pastry presents good scaling properties, as its routing state increases logarithmically with the size of the overlay. In addition, OARs are expected to present lower churn rates compared to end-hosts, where Pastry normally resides, thus further reducing routing information exchange overhead. Second, the Scribe overlay multicast scheme requires additional forwarding state, further augmented by MultiCache specific state, i.e., the AS number of a joining OAR and a bitfield for monitoring content delivery (see Sections II-C and II-D1). In this section we investigate the scalability properties of MultiCache, in order to assess the amount of resources required to support it. To this end, we study two important aspects of the load imposed on OARs, namely, the structural characteristics of the Scribe overlay multicast trees, which dictate the distribution of the forwarding load, and the size of the imposed workload. Table I provides a summary of the notation employed hereafter.

### A. Overlay multicast tree properties

The impact of deployment density on the resulting multicast trees is expressed through the size of the overlay, i.e., the total number of deployed OARs, $N_o$. To simplify our analysis, we assume that all AS's adopt MultiCache and focus on the intradomain deployment density ($d$), assumed to be the same across all AS's. Hence, we have $N = \sum_{i=0}^{D-1} R_i$ and $N_o = dN$. We also assume that end-hosts are uniformly dispersed in the network and that requests submitted to the same OAR for the same data are aggregated and, ultimately, served via a single subscription. Following [23], we express the probability of the shortest overlay distance between two Pastry nodes being $i$ hops with a binomial distribution $L_i$, where $p = \frac{1}{2^b}$ and $b$ is the Pastry configuration parameter:

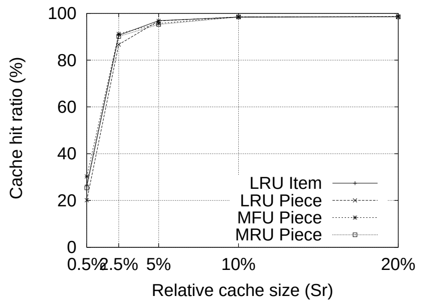
| | |
|---|---|
| $U$ | Number of end hosts |
| $N$ | Number of access routers |
| $N_o$ | Number of overlay access routers |
| $D$ | Number of domains |
| $R_i$ | Number of access routers in domain $i$ |
| $d$ | Deployment density |
| $L_i$ | Probability Mass Function of shortest overlay distance in Pastry |
| $b$ | Pastry configuration parameter |
| $C_i^j$ | Number of end host requests for item $j$ at tree level $i$ |
| $J_i$ | Average number of forwarding links at tree level $i$ |
| $K_i$ | Average number of forwarding state entries per node at tree level $i$ |
| $M$ | Number of files distributed |
| $P$ | MultiCache piece size (MB) |
| $Q$ | Average file size (MB) |

TABLE I
Analysis notation

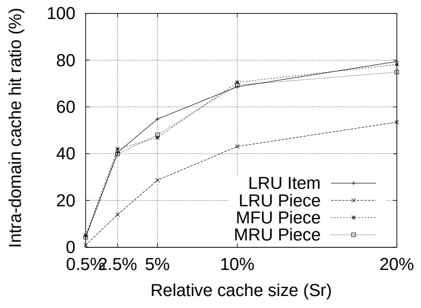
$$L_i = \binom{\log_{2^b} N_o}{i} p^i (1-p)^{\log_{2^b} N_o - i} \tag{1}$$

If $C$ is the total number of end hosts that request a certain item, we calculate the total number of leaves $C_i$ that are $i$ hops away from the root of the respective tree as follows:

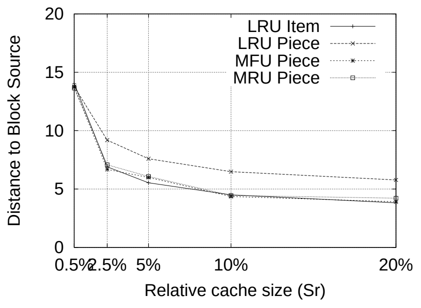
$$C_i = min[CL_i, N_o L_i] \tag{2}$$



(a) Cache hit ratio (CHR)

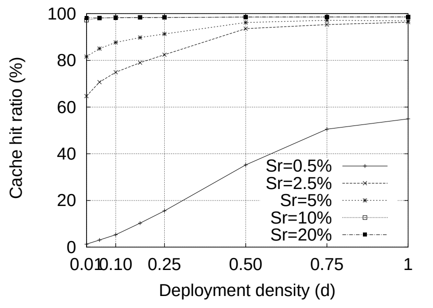


(b) Intradomain CHR (CHR-Intra)

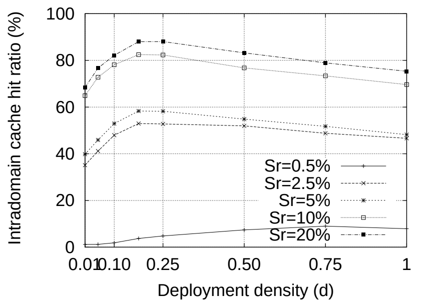


(c) Distance to block source (DBS)

Fig. 4. Effect of cache replacement policies on MultiCache ($d = 0.25$).



(a) Cache hit ratio (CHR)



(b) Intradomain CHR (CHR-Intra)

Fig. 5. Effect of deployment density on MultiCache ($MFU$).

This equation expresses the aforementioned aggregation of similar requests at proxy OARs. Then, we calculate the number of forwarding links $J_i$ at each level $i$ of a multicast tree, by considering the number of leaf nodes at each level of the tree, as well as the number of forwarding links towards leaf nodes residing at lower levels. Two forwarding links from level $i$ to $i+1$ of a multicast tree stem from the same node at level $i$ with probability:

$$S_i = 1 - (1 - \frac{1}{N_o \cdot L_i})^{J_i} \tag{3}$$

This equation expresses the portion of forwarding links merging at level $i$ nodes, that is, at each level $i$ of the tree, $S_i J_i$ links merge on some node(s) at that level. At the one extreme, these links may all merge at a single node (hereafter called the *MaxMerge* case), thus incurring a single forwarding link at level $i-1$. At the other extreme, they may merge in pairs incurring $\frac{S_i J_i}{2}$ forwarding links at level $i-1$ (hereafter called the *MinMerge* case). Fig. 6, illustrates both cases. We calculate $J_i$ beginning from the lowest level of the tree, where all forwarding links lead to leaf nodes. Then, traversing the tree towards the root, we take the two boundary merging cases and calculate the number of forwarding links that are required for the data to reach the lower, already visited, levels of the tree. At each level $i$, part of these forwarding links passes through non-leaf, forwarding nodes at level $i+1$, and the remainder passes through leaf, forwarding nodes of that level, i.e., nodes that both consume and forward the received data. Additional forwarding links are also considered for all leaf nodes of the next level that do not act as forwarders. Hence, $J_i$ is calculated as follows:

$$J_i = F_i(1 - \min[\frac{C}{N_o}, 1]) + C_{i+1} \tag{4}$$

$$F_i = \begin{cases} (1 - S_{i+1})J_{i+1} + 1 & , \textit{MaxMerge} \text{ case} \\ (1 - S_{i+1})J_{i+1} + \frac{S_{i+1}J_{i+1}}{2} & , \textit{MinMerge} \text{ case} \end{cases} \tag{5}$$

Next, we calculate $K_i$, the average number of children entries maintained at a level $i$ OAR, assuming that the nodes residing

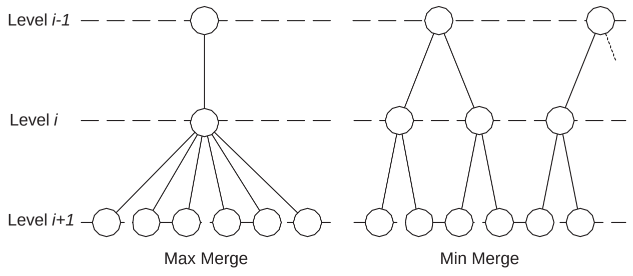


Fig. 6. Boundary merging cases.

at each level of the multicast tree share equally the forwarding links to the next level of the tree.

$$K_i = \begin{cases} \frac{J_i}{J_{i-1}} & , i > 0 \\ J_i & , i = 0 \end{cases} \tag{6}$$

### *B. Workload impact*

Based on the properties of the created overlay multicast trees, we first determine the workload imposed by the end users and then calculate the amount of forwarding state required per OAR to support it. To this end, we focus our analysis on a simple content distribution scenario where a MultiCache enabled network distributes $M$ files to the end hosts of the network. We simplify our analysis with the following assumptions:

1) All delivered files have equal size $Q$.
2) Each file is fragmented into pieces of size $P$ and $Q\%P = 0$, i.e., for each distributed file exactly $\frac{Q}{P}$ multicast trees are formed.
3) We study the properties of the resulting trees once all recipient OARs for each file have joined the respective multicast trees.

The size of the resulting workload depends on file popularity, which ultimately determines the size of the resulting delivery trees. We model the popularity of files $j \in [1, M]$, with a Mandelbrot-Zipf distribution $P(j)$ [14]. We normalize this distribution so that the most popular item is requested by all end hosts ($U$), and calculate the total number of requests per file $j$:

$$C^j = \frac{P(j)}{P(0)} \cdot U \tag{7}$$

Note that the actual number of leaf nodes is determined by deployment density (see Equation 2). Moreover, due to content fragmentation, $C^j$ translates to an equal number of requests for each of the $\frac{Q}{P}$ resulting pieces.

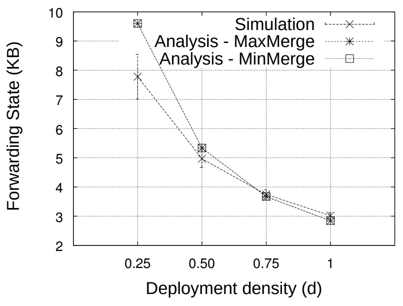
Based on the above, we can calculate the forwarding state load for each tree by first deriving the total number of requests and then employing Equations 1 to 6. Note that, on the average, all OARs may act as forwarders for each tree with equal probability. Additionally, a forwarding OAR is equally likely to reside at any level of the respective tree. Hence, we evenly distribute the resulting load to all $N_o$ OARs in the network. Fig. 7(a) presents the forwarding state required per OAR, for a sample scenario with $N = 2400$, $U = 1500$, $M = 500$, $Q = 16$ MB, $P = 16$ MB, derived via both analysis and simulation. The agreement between the two methods is notable, especially in denser deployments. For sparser deployments, the analysis overestimates the forwarding state required.

Due to simulator limitations, we turn to the analytical model to investigate the scaling properties in the case of much larger topologies and workloads. Figure 7(b) presents the aggregate forwarding state at each OAR, for various workload sizes and deployment densities. In the scenario depicted, we have considered a network of 15,000 access routers with 10,000 uniformly dispersed end-hosts. The various workload sizes refer to files of size $Q = 656$ MB, i.e., close to the median file size (651 MB) observed in [21], and $P = 16$MB. We consider a 256-bit memory footprint for each forwarding entry, consisting of the 128-bit Pastry ID of the target OAR along with its 32-bit IP address, the 32-bit AS number and a 64-bit *bitfield* for monitoring the forwarded content (considering 256 KB blocks). Note that keeping track of cache availability indicated by CACHE UPDATE messages does not require additional state, as this information can be marked on the *bitfield*. The results shown are the average of the *MaxMerge* and *MinMerge* cases. We can see that even in the highest workload case, the memory footprint for forwarding state does not exceed 5 MB, demonstrating the scalability of MultiCache.

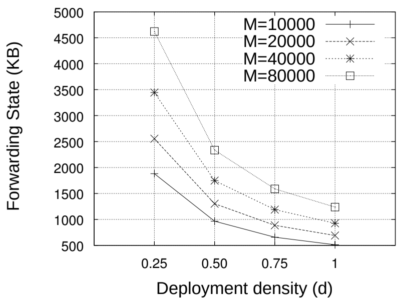
## V. RELATED WORK

Information centric networking has caught the attention of the research community during the past few years (e.g., [9], [24], [25]). The PSIRP Project [9] follows a clean-slate approach, proposing a network architecture based on the Publish/Subscribe paradigm. The end-to-end principle is replaced by a flexible and expressive rendezvous system that acts as a mediator for locating information, combined with source routing based on *zFilters* i.e., a variant of Bloom filters used to compress delivery paths inside packet headers [26]. As PSIRP considers a network architecture unconstrained by existing Internet functionality, it raises important deployment concerns. MultiCache attempts to overcome these concerns by following an evolutionary approach, building on top of the existing architecture in an overlay fashion.

The *Content Centric Networking* (CCN) architecture [24] also aims at replacing the end-to-end, conversational model of the current Internet architecture, focusing instead on information. In CCN, content is requested with `Interest` packets that flood the network, leaving reverse path information at crossed routers. `Data` packets follow these trails in order to reach the origin of the request. A hierarchical structure is employed to represent the content name space in a scalable manner. CCN can be incrementally deployed on top of IP, as is the case of MultiCache. However, there are scalability concerns when considering inter-domain level routing and forwarding, stemming from the enormous size of the content namespace, which have not been tackled yet. In MultiCache, these concerns are addressed with the logarithmic properties of the underlying Pastry DHT and the demonstrated scalability of Scribe which efficiently distributes multicast forwarding information [2]; we further investigated this issue in Section IV.



(a) Analysis vs. Simulation



(b) Analysis: high workload

Fig. 7. Forwarding state per OAR

MultiCache shares some important design features with the DONA architecture [25]. The latter builds an overlay content centric layer based on *Resolution Handlers* (RH) which are responsible for content centric routing, allowing nearby locations of the desired content to be located via anycast. Unlike DONA, MultiCache focuses on resource sharing and further delves into the details of the data delivery plane, enabling the joint provision of multicast and caching. In MultiCache, deploying multiple OARs inside an administrative domain becomes the norm, allowing the existence of multiple caching locations, this enabling traffic localization and improving end-user experience. Unlike MultiCache, DONA only allows clients to benefit from caching locations on the path towards the root level of the RH hierarchy.

The *Cache-and-Forward* (CNF) architecture follows a similar approach, by employing content-based routing on top of IP and providing caching services by in-network devices [27]. However, it is hard to engage in a detailed comparison with MultiCache, as the architecture is still shaping, with some major design decisions and features not yet finalized (e.g., name resolution, multicast) and only a preliminary set of simulation results available.

At the application level, SplitStream [13] stripes content to produce a forest of disjoint Scribe trees in order to spread 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 [28].

The excessive amount of traffic generated by P2P applications has motivated considerable research (e.g., [1], [29]). Often, co-operation between ISPs and P2P applications is proposed to allow the former to provide rich underlay information to the latter, so as to improve the decisions made during peer selection. Though reducing the consumption of network resources and localizing traffic, these approaches maintain the current end-point centric paradigm, providing a P2P specific solution. In contrast, MultiCache attempts to establish a radically different networking paradigm, where the network focuses on information rather than on end-hosts, multicast and caching are the norm and the usage model is substantially simplified.

## VI. Conclusions and future work

The information-centric networking paradigm has drawn the attention of the research community, due to its ability to express current network usage patterns, which have shifted from the traditional conversational model. In this paper, we have presented MultiCache, an overlay network architecture realizing the information centric model. The overlay character of this approach targets an incremental, evolutionary transition process, enabling the gradual deployment of the proposed functionality. In our architecture we build on information-awareness for the joint deployment of resource sharing mechanisms such as caching and multicast. Simulation results have demonstrated the considerable benefits of the proposed architecture compared to the BitTorrent application. By investing in even sparse deployments of MultiCache, network operators can regain control of network traffic, significantly reducing the load imposed on their infrastructure, while at the same time providing their end-users with substantially reduced download times.

Our next steps focus on improving the adaptation of the MultiCache architecture to the structure of the current Internet. Our target is to depart from our flat overlay design, so as to better reflect the relationships between AS's in MultiCache. To this end, we have designed a hierarchical version of Pastry, based on the Canon paradigm [30], that can provide network operators with finer control of interdomain service provision.

## VII. Acknowledgements

The work reported in this paper was supported by the EU FP7 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 2005 ACM/USENIX IMC*, 2005, pp. 1–6.
[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 2001 Middleware Conference*, 2001, pp. 329–350.
[4] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.
[5] K. Katsaros, N. Bartsotas, and G. Xylomenos, "Router assisted overlay multicast," in *Proc. of the 2009 Euro-NGI Conference on Next Generation Internet Networks*, 2009, pp. 329–350.

[6] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, “Handling churn in a DHT,” in *Proc. of the 2004 USENIX ATEC*. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.
[7] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, “Incentive-compatible caching and peering in data-oriented networks,” in *Proc. of the 2008 ACM CoNEXT*. New York, NY, USA: ACM, 2008, pp. 1–6.
[8] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, “Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination,” in *Proc. of the 2001 ACM NOSSDAV*, 2001, pp. 11–20.
[9] PSIRP Project, *PSIRP Home Page*, http://www.psirp.org, 2010.
[10] IANA, “Autonomous System (AS) Numbers,” http://www.iana.org/assignments/as-numbers/as-numbers.xml, 2010.
[11] I. Smith, “Historical notes about the cost of hard drive storage space,” http://ns1758.ca/winch/winchest.html, 2010.
[12] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, “Scalable application-level anycast for highly dynamic groups,” in *Proc. of the 2003 Networked Group Communication Workshop*, 2003, pp. 47–57.
[13] 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 2003 ACM SOSP*, 2003, pp. 298–313.
[14] M. Hefeeda and O. Saleh, “Traffic modeling and proportional partial caching for Peer-to-Peer systems,” *IEEE/ACM ToN*, vol. 16, no. 6, pp. 1447–1460, 2008.
[15] C. Diot, B. N. 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.
[16] A. Varga and R. Hornig, “An overview of the OMNeT++ simulation environment,” in *Proc. of the 2008 ICST SIMUTools*, 2008, pp. 1–10.
[17] I. Baumgart, B. Heep, and S. Krause, “OverSim: A flexible overlay network simulation framework,” in *Proc. of the 2007 IEEE GI Symposium*, 2007, pp. 79–84.
[18] K. Katsaros, V. Kemerlis, C. Stais, and G. Xylomenos, “A BitTorrent module for the OMNeT++ simulator,” in *Proc. of the 2009 IEEE MASCOTS*, 2009, pp. 361–370.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] V. Pappas, D. Massey, A. Terzis, and L. Zhang, “A comparative study of the DNS design with DHT-based alternatives,” in *Proc. of the 2006 IEEE INFOCOM*. IEEE, 2006, pp. 1–13.
[24] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in *Proc. of the 2009 ACM CoNEXT*. New York, NY, USA: ACM, 2009, pp. 1–12.
[25] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” in *Proc. of the 2007 ACM SIGCOMM*. New York, NY, USA: ACM, 2007, pp. 181–192.
[26] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: Line speed publish/subscribe inter-networking,” in *Proc. of the 2009 ACM SIGCOMM*. New York, NY, USA: ACM, 2009, pp. 195–206.
[27] L. Dong, H. Liu, Y. Zhang, S. Paul, and D. Raychaudhuri, “On the cache-and-forward network architecture,” in *Proc. of the 2009 IEEE ICC*, jun. 2009, pp. 1 –5.
[28] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, S. Seshan, and H. Zhang, “The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols,” in *Proc. of the 2005 IPTPS*, 2005, pp. 115–126.
[29] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, “P4P: Provider Portal for Applications,” in *Proc. of the 2008 ACM SIGCOMM*. New York, NY, USA: ACM, 2008, pp. 351–362.
[30] P. Ganesan, K. Gummadi, and H. Garcia-Molina, “Canon in G Major: Designing DHTs with Hierarchical Structure,” in *Proc. of the 2004 ICDCS*, 2004, pp. 263–272.