[Article] Design and implementation of locality-aware P2P system

Design and Implementation of a locality-aware P2P system

Luigi Ciminiera, Guido Marchetto, Marco Papa Manzillo, Vinicio Vercellone, Mario Ullio

Author's version
Published in
Recent Patents on Telecommunications, vol. 1 n. 1, pp. 23-32
The final published version is accessible from here:
http://dx.doi.org/10.2174/2211740711201010023

Abstract

One of the most relevant problem for an Internet Service Provider is the large bandwidth usage on international links, mainly due to peer-to-peer applications adopted for file-sharing. The Collaborative Locality-aware Overlay SERvice (CLOSER) technology has been recently proposed to solve this issue by properly modifying the behavior of peer-to-peer application. The technology is also covered in two recent patent applications. This paper presents possible design guidelines to actually implement CLOSER in a DHT-based peer-to-peer system and describe a real implementation based on the popular aMule application.

I. Introduction

The problem of traffic locality has significantly grown in importance during the last few years. Especially due to the wide spread of peer-to-peer (P2P) technologies for file-sharing, the total volume of traffic crossing expensive international links is steadily increasing, with consequent raise of costs for Internet Service Providers (ISPs). For these reasons, ISPs are actively looking for effective solutions that aim at segregating P2P traffic within their boundaries without affecting, or rather improving, users' experience. A simple example of locality-awareness is depicted in Figure 1. The image shows a small system with the locality-awareness exploited (right side) and not exploited (left side). Triangles represent peers that are downloading the same resource, while squares represent peers that share such resource. In this trivial example, the traffic becomes totally circumscribed in the ISPs infrastructures.

Among many existing solutions [1]–[5] the Collaborative Locality-aware Overlay SERvice (CLOSER) has been recently proposed [6] and some of its principles covered in two patent applications [7], [8]. The main improvement that CLOSER offers with respect to other solutions comes from two principles: the use of the indexing system to store and manage localization data, in conjunction with the acquisition of such information at registration time. This allows considering all available peers when selecting the ones to contact for download, which is not possible with other approaches and clearly improves the system performance from a traffic locality perspective. Furthermore, CLOSER implements a distributed caching system (specifically covered in [8]) that favors the increase of local copies of a given resource, thus potentially reducing both the probability that a resource is downloaded from the outside of the ISP and the average download time experienced by users. This module can also provide users' privacy in an efficient and scalable way, which is one of the most relevant issues in current P2P networks.

This paper presents a possible realization of CLOSER, discussing both the traffic locality aspect and the content selection procedure (i.e., the algorithm adopted for the selection of the resource to download) of the distributed caching system. In essence, we propose a CLOSER-aware P2P system based on a DHT and we present an algorithm for proactive caching that favors the download of interesting content. Furthermore, the paper describes in detail a real software implementation of CLOSER, based on the aMule [9] P2P application. In essence, while [6] provides a general description of CLOSER and its operating principles, this paper covers more practical aspects of the approach, concerning both traffic locality and caching subsystems.

The paper is organized as follows. Section II provides some background information on CLOSER, also highlighting the main strengths of the approach with respect to other existing solutions. Section III presents a DHT-based realization of CLOSER, also

L. Ciminiera, G. Marchetto, and M. Papa Manzillo are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy (email: name.surname@polito.it). V. Vercellone and M. Ullio are with Telecom Italia Labs, Via Reiss Romoli 274, Torino, Italy (email: name.surname@telecomitalia.it.

considering a possible solution for the distributed caching module, while Section IV describes our CLOSER-aware P2P system implementation. Finally, Section V concludes the paper.

II. Background

A. Existing solutions

Promising approaches to achieve traffic locality in P2P networks are based on the modification of some components of P2P systems, possibly in conjunction with the deployment of additional modules that slightly modify current P2P paradigms.

A first approach consists in modifying the behavior of current P2P applications, so that they can autonomously acquire their localization information and provide it to other users interested in the resources they share. Ono [2], a software extension of the Azureus BitTorrent client, is an example of these systems. In Ono, each P2P client determines its location by properly querying a Content Delivery Network (CDN). A similar solution is Kontiki [3], where clients acquire their localization information by retrieving their AS Number (ASN) — assigned to ISPs by the Internet Assigned Numbers Authority (IANA) — from public databases.

A second approach is based on the deployment of proper servers that the indexing system or P2P clients by themselves (it depends on the specific solutions) can query to obtain the localization information of peers providing a given resource (referred in the following to as *resource providers*). Main examples are the oracle [4], P4P [5], and ALTO [10]. For example, the P4P/ALTO solutions propose to deploy a central server that the indexing system contacts before sending the list of resource providers to a querying user in order to acquire the localization information of these providers. The *oracle* has a similar functionality in the solution proposed in [4], but is queried by users after they receive the list of possible resource providers.

All these solutions have a common operating principle: the locality information related to the resource providers is acquired (either by the applications or by the indexing system) at *lookup time*, i.e., when a user starts a lookup for a given resource. In Ono, Kontiki, and the oracle-based solution, users acquire a list of resource providers from the indexing system, and then collect locality information related to the listed providers. However, for scalability reasons, indexing systems generally do not supply nodes with an exhaustive list of resource providers, but randomly selects a subset of L resource providers among all the available ones — by default, $L = 50$ in BitTorrent. Let us denote this list as *sampled list*. Since every ISP includes a small percentage of the Internet population, it is unlikely that the sampled list includes a high number of resource providers located in the same ISP. Hence, the optimization process executed by these techniques may not be very effective. A similar problem is present in P4P/ALTO, as the indexing system can send only a "sampled list" of the available resource providers to the central server providing localization data, which hence performs a suboptimal ranking. We denote this issue as *sampled list problem*.

Furthermore, the oracle technique, P4P, and ALTO need a massive intervention of the ISP, which is required to deploy and maintain the equipment (the central server) for ordering the list of possible service providers. It is also worth noticing how the presence of such equipment may allow malicious users to reconstruct the ISP topology — that generally is a confidential information — by forging ad hoc requests and analyzing the server answers. On the other side, Ono and Kontiki do not require the ISP intervention and are less sensible to malicious peers aiming at reconstructing the ISP topology, but to the detriment of the localization precision.

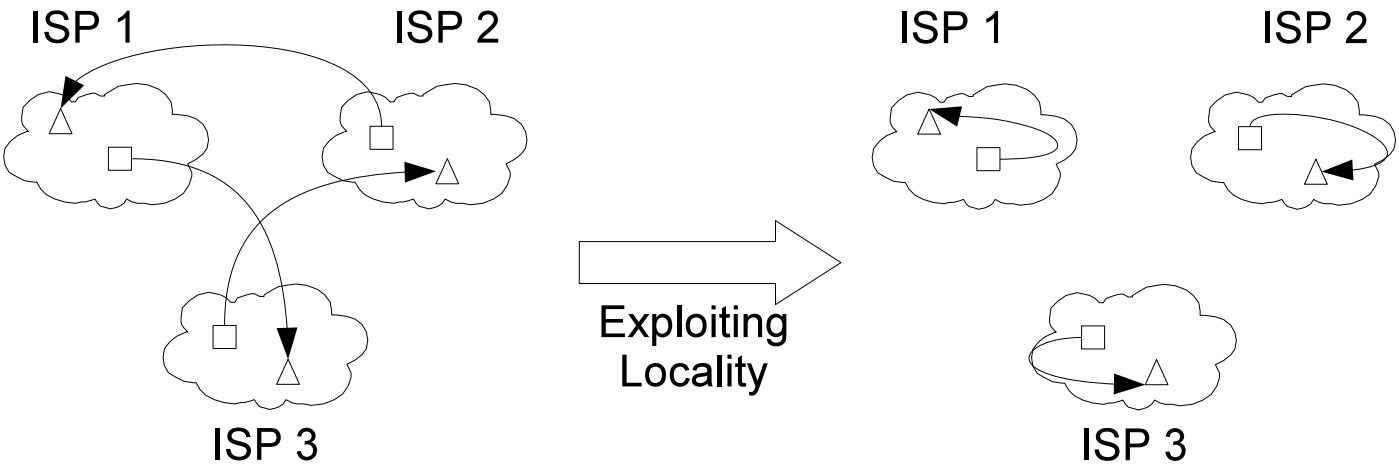

Fig. 1. A simple example of locality-awareness.

B. CLOSER overview

CLOSER locality-awareness relies on a simple idea: the indexing system is made aware of the localization information of every resource provider. This is done by enabling resource providers to communicate their localization information to the indexing system whenever they register a new resource, i.e., at *registration time*. During a lookup procedure, the requester gives its own localization data to the indexing system, which sorts the entire resource provider list by increasing distance from it. Then, the indexing system forwards the list to the requester, which, consequently, is enabled to download from the closest peers. In essence, even if the indexing system has to limit the resource list to C entries for scalability reasons, the first C entries are the most interesting from the locality-awareness point of view. Thus, unlike the above described systems, CLOSER avoids the sampled list issue.

To support CLOSER, ISPs do not have to deploy any infrastructure: they simply have to provide nodes with their localization data, which can be easily distributed through widely used systems — e.g., a web application. This is another advantage with respect to other techniques such as the oracle-based or P4P, which instead have to maintain specific servers. Furthermore, in CLOSER the ISP provides the localization information to each single resource provider and, thus, a malicious user should acquire localization data from every single user to reconstruct the ISP topology. This may be complicated as users' applications are not programmed to reply to direct queries concerning their topological information.

CLOSER also includes a users' privacy module which also operates as a distributed and proactive caching system: P2P applications automatically select and download resources, even if those are not requested by the user. This approach offers users' privacy because, observing a node behavior, it is hard to determine if resources were requested by the actual user or by an automatic download process. This solution overcomes traditional privacy provision approaches (e.g., [11], [12]), which are based on an intensive use of proxies and hard cryptography and consequently offer poor performance from a download time perspective. These features are detailed in [6], which however omits considering the distributed caching functionality of the module. This will be discussed in Section III-B, which also describes a specific algorithm for content selection.

Analytical and simulation results [6] demonstrated the effectiveness of CLOSER in providing traffic locality, also showing how it mimics an ideal locality-aware P2P system. Furthermore, CLOSER and its distributed caching system are object of two patent applications [7], [8].

III. System design and evaluation

Given the operating principles of CLOSER, briefly summarized in the previous section, we present possible design guidelines to actually define the CLOSER architecture, both considering traffic locality and distributed caching aspects.

A. DHT-based CLOSER realization

The success of any strategy is strictly related to its adoption by users. Thus, it is very important that the solution is such appealing for the user community to encourage the P2P application switching. The traffic locality may not be sufficient to guarantee a wide adoption because users do not generally pay connections according to the distance with the counterpart. For this reason it is necessary to take into account aspects that are not strictly related with traffic locality but are of interest from the user perspective. This means that a novel solution should follow historical P2P trends and, at the same time, should not require a clean-slate approach; indeed, it is necessary to inherit the customer base of a popular P2P system, as the number of available resources is a key factor for the success of a file-sharing application. In essence, our application has to ensure backward compatibility with any popular P2P system. While this is trivial in the server based indexing system context (e.g., CLOSER can be implemented as an extension of BitTorrent trackers), it requires investigation when the indexing system is based on a Distributed Hash Table (DHT). Hence, we present here a DHT-based CLOSER realization.

1) The local DHT: In our system design, the CLOSER topology consists of a first legacy DHT network, e.g. Kad, serving worldwide users, and a second DHT for each Autonomous System (AS). The latter is used as an index only by users belonging to the same AS.

The ISP provides its users with their localization data, sending three pieces of information, as described in [6]: (*i*) the AS Number (ASN), (*ii*) the Area ID, and (*iii*) the POP ID. AS number and POP ID are self explaining, while Area ID defines a sub zone that groups several POPs of the same ISP; zones may be useful for both large ISPs and small organizations (that do not form an AS).

Peers share their resources on both the legacy system and their local DHTs. The registration on the legacy system is performed as usual. Instead, in the local DHT, peers also register their Area and POP ID (the AS is implicit).

When a peer wants to retrieve a resource, it executes a first search in its local DHT sending its Area and POP ID. The local DHT builds a sorted list of results. On top of the list, it inserts the resource providers with the same Area and POP ID as the requester, followed by providers with the same Area ID and, finally, the others that, however, belong to the same AS as the requester. If the requester does not obtain enough providers, it executes a second search in the legacy system.

In this way, the system achieves locality-awareness by implementing CLOSER. This mechanism also allows inheriting the customer base of the legacy system, because CLOSER enabled peers may coexist with older ones. In addition, it is possible

to discover *all* the resource providers placed close to the requester since the AS indexing systems are based on a DHT. It is interesting to notice that, by definition of DHT, the AS indexing system is able to point out all the local resources. Thus, if all nodes use CLOSER, the locality-awareness achieved is ideal. Moreover ISPs are not obliged to take part to the indexing system. This has various benefits:

1) ISPs do not require equipment devoted to this role
2) Changes to the protocol do not require ISP support
3) There are not legal concerns for ISPs
4) Users are not concerned about potential profiling conducted by ISPs.

2) ISP confederations: The above mechanism also allows CLOSER to support the creation of ISP confederations. If two or more ISPs have business agreements for which CLOSER should consider them as a unique zone, it is sufficient for them to provide users with the IDs of the confederated ISPs. When querying for a resource, users communicate to the indexing system also this list, so that it can give priority to the confederated ISPs if the resource is present both within and outside the confederation. Optionally, different priorities can be assigned to different ISPs in the confederation (e.g., to preferably select ISPs that can be reached through high capacity links). These have to be communicated to the users by the ISP together with the above mentioned list, so that they can use it when interacting with the indexing system to locate a resource.

Confederations may also further reduce the Tier-1 transit traffic, as shown in Figure 2: without confederations, a user in the ISP A may retrieve a given resource from the ISP B as well as from a generic ISP N, thus possibly using the Tier-1 ISP; on the other hand, the support that CLOSER offers to confederations allows ISP A to favor downloads from the ISP B, thus exploiting their peering agreement. Moreover, the ISP could optionally specify different confederation IDs, according to the type or the size of resources a user is interested in sharing or retrieving. For example, an ISP A could notify its users of the existence of three different confederation systems: the confederation A ‖ B for audio files of whatever size, the confederation A ‖ C ‖ D for video files whose size is up to 100MB, and the confederation A ‖ D for video files greater than 100MB. This feature can increase the ISP flexibility and could open the path for the definition of new business models and new balances in the relations among ISPs.

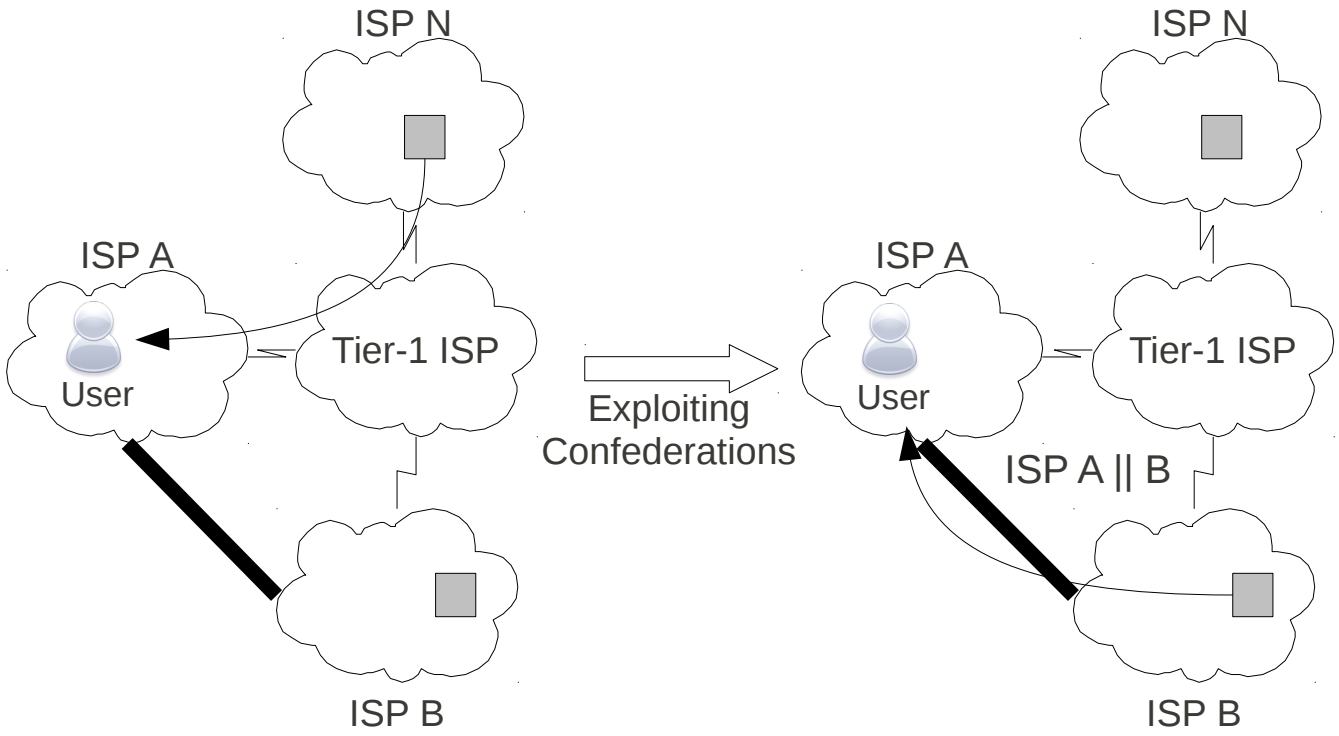

Fig. 2. Benefits of confederation system.

B. Distributed caching system

CLOSER also includes a mechanism for automatic downloads of resources, which in [6] is mainly described as a privacy module: observing a node behavior, it is hard to determine if resources were requested by the actual user or by an automatic download process, thus confusing possible eavesdroppers.

From this point of view, resources for automatic downloads can be selected randomly. However, automatic downloads can be exploited to possibly reduce the download times experienced by users, as depicted in one of the patent applications regarding CLOSER [8]. In particular, if automatic download instances fetch the resources that are likely to be requested in the near future by users belonging to a given ISP, the users' privacy system can also offer benefits similar to a distributed proactive caching. In essence, the presence of more copies of a critical resource within the ISP boundaries can improve the performance of the swarming and, at the same time, preserve traffic locality as users will have more opportunities to download that resource without involving external resource providers.

1) Criticality index: Given the above reasons, we define a heuristic algorithm for the evaluation of the importance for an ISP to have a given resource within its boundaries from a user perspective (i.e., how likely that resource will be requested in the near future), denoted in the following as the *criticality index* $I_c(i)$ of a resource i in the considered ISP. The heuristic is based on these simple principles: (*i*) the larger the number of active downloads of a given resource i from the interior of the ISP, the greater is the importance for the ISP to increase the number of internal copies of the resource, but (*ii*) the greater the number of internal copies of i, the higher is the performance of the swarming for that resource and, consequently, the lower becomes the necessity for the ISP to further increase the number of internal copies of i.

Clearly, once a downloading node has retrieved a chunk of the resource i, it can immediately offer it to other downloading nodes which still miss this particular chunk. Assuming the chunks to be uniformly distributed in the network, we can argue that a node owning a percentage p of a resource i can satisfy a percentage p of the requests for that resource. Hence, these nodes have to be considered, weighted with p, in the computation of the number of resources available within the boundaries of the ISP. To avoid the scalability issues deriving from considering every single chunk in this evaluation, we define five discrete levels of download completion (i.e., 0%, 25%, 50%, 75%, and 100%). Assuming an average download completion $\frac{(p_a+p_b)}{2}$ for all downloads whose percentage of completion is between p_a and p_b we have five possible weights p (i.e., 0.125, 0.375, 0.625, 0.875 and 1). Let $N_{AD}(i)$ denote the number of active downloads of the resource i (excluding automatic downloads as these are themselves driven by the criticality index and hence have not to influence its evaluation for stability reasons) in a given ISP, $N_{RP}(i)$ denote the number of resource providers located within the boundaries of the ISP, weighted with their download completion percentage, and $N_{p\%}(i)$ denote the number of nodes providing at least a percentage p of the resource. For a resource i we have:

$$I_c(i) = \begin{cases} 0 & \text{if} \quad N_{AD}(i) < \bar{k} \\ \left(\frac{N_{AD}(i)}{N_{RP}(i)}\right) \cdot \frac{1}{8} & \text{otherwise,} \end{cases} \tag{1}$$

$$\begin{aligned} N_{RP}(i) &= N_{100\%}(i) + N_{75\%}(i) \cdot 0.875 + \\ &+ N_{50\%}(i) \cdot 0.625 + N_{25\%}(i) \cdot 0.375 + \\ &+ N_{0\%}(i) \cdot 0.125. \end{aligned} \tag{2}$$

I_c is set to zero if the number of requests is lower than a threshold $\bar{k}$ for avoiding automatic downloads of unpopular resources. The factor $\frac{1}{8}$ is used to normalize the criticality index, as $\frac{N_{AD}}{N_{RP}}$ assumes values that are always between 0 (number of requests negligible) and $\frac{1}{0.125} = 8$ (when all downloads are in 0%-25% range).

The criticality index of each resource is evaluated by the indexing system[1]. In particular, each peer notifies the indexing system whenever it reaches one of the download completion levels, so that it can maintain the value of N_{RP} up-to-date. Furthermore, the indexing system estimates N_{AD} by increasing this value whenever a new request for a given resource arrives and decreases the value whenever either a node notifies the download completion for that resource or a timeout expires at the indexing system concerning that download (this means that the node left the overlay). However, an issue may raise as N_{AD} has to consider only real downloads: P2P applications have to notify if their download requests are user-driven or automatic, thus enabling an eavesdropper acting as indexing system to break users' privacy. This can be avoided by borrowing the concept of proxy-based communications from existing privacy solutions: whenever users interact with the indexing system, they use a traditional anonymity system based on proxies (e.g., [11], [12]), so that the eavesdropper can discriminate between real and automatic downloads but cannot discover the origin of the requests. Notice that, since data exchanged with the indexing system generally does not require a high bandwidth because are relatively small, such policy does not affect the system performance; downloads, which instead are bandwidth intensive, do not involve proxy nodes, thus preserving this important feature introduced in [6] concerning the privacy module.

Whenever a resource request arrives at the indexing system, this includes in its reply the criticality index of the requested resource and of a set of 2-uples <resource ID, I_c> randomly selected among the ones it knows. This allows the querying peer to learn these 2-uples and use them to drive CLOPS downloads. Analogously, whenever an interaction occurs between two nodes to start a download, those share a subset of the criticality indexes they know. In essence, a gossip mechanism

[1]This evaluation can be done in both centralized and distributed indexing systems. In the former case, the centralized server handles the criticality index of all the available resources, while in the latter each node evaluates the criticality index of the subset of resources it is responsible for. In the following we generically refer to indexing system for simplicity.

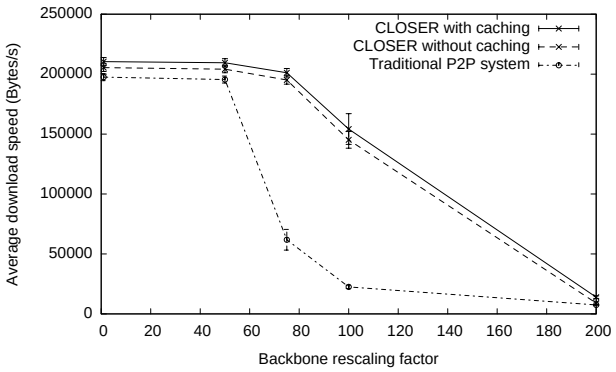

Fig. 3. Average download speed with traffic shapers on backbone links.

is implemented, which enables nodes to learn and possibly update criticality indexes and hence profitably perform automatic downloads: all instances collect the criticality indexes and download the resource associated to the higher index. Since resources may have different criticality in different ISPs, the indexing system evaluates and provides criticality indexes specific for each involved ISP. For the same reason, gossiping is enabled only among nodes of the same ISP. Notice that this is favored using CLOSER as download connections are usually established among nodes belonging to the same ISP.

2) Simulation results: Although the following section introduces a real CLOSER implementation, the evaluation of this specific aspect of the system architecture is done by simulation as it is necessary to involve a large number of nodes, which is impractical in a real testbed. However, the network of prominent Italian ISP is reproduced within the simulator in order to obtain significant results of practical interest. Furthermore, we adopt a Poisson distribution for both the session lengths and the user arrivals. We set the average session length to 40 hours, according to what reported in [13]. Furthermore, we set the average arrival rate to 1 user per second, which coupled with the above session length results in an average overlay population of 144,000 nodes in the steady state. The number of resources shared by users is generated by adopting a Weibull distribution with scale $= 42$ and shape $= 0.5$, as depicted in [13]. The selection of the resources to share or request is modeled with a Zipf-Mandelbrot distribution, as proposed by in [14]. Finally, we assume resources are uniformly distributed among ISPs.

Figure 3 depicts the average download speed[2] as a function of the bandwidth fraction available for P2P traffic. We can observe how download speed is almost constant for small backbone rescaling factors. Furthermore, the presence of CLOSER and its distributed caching system does not give significant benefits in this situation as the available bandwidth is sufficient to efficiently support resource downloads. However, it is worth noticing how our system significantly outperforms the traditional P2P paradigm (i.e., without CLOSER and its caching module) with a restrictive backbone throttling, which may be a common situation in nowadays networks.

IV. System implementation

The guidelines introduced in the previous section are used to implement a real CLOSER-aware P2P application. We derive our application from the well known C++ aMule client [9], based on the Kad network (i.e., an implementation of the Kademlia DHT [15]). We consider this P2P client a good choice for our purpose given that it is a popular system and adopts a totally decentralized DHT.

The rest of this section is organized as follows. First, we briefly present the Kad architecture, then we describe the modifications required to add CLOSER functionalities. Finally, we present some experimental results obtained over the PlanetLab infrastructure.

A. The Kad network

Kad is an implementation of the Kademlia protocol, supported by the aMule and eMule clients. The main differences from the original protocol are due to the backward compatibility with the eDonkey protocol.

The main tasks of the Kad protocol are handled by the class `CKademlia`, which controls the instances of the other classes required to perform all the operations made by a peer. This class is singletone: it is possible to instantiate just one instance by using the static method `CKademlia::Start()` and delete it with the static method `CKademlia::Stop()`. With regard to the management operations, they are periodically executed by the static method `CKademlia::Process()`, called by a timer.

Other important classes for the Kad network management are:

[2]We preferred to measure the average download speed instead of the download time (its more common symmetric quantity) to avoid issues related to incomplete downloads due to users leaving the system.

- `CPrefs`: it contains the peer details required for the DHT operation, such as Id, IP address, and port and status information;
- `CRoutingZone`: it is used to store and manage the routing table of the overlay, i.e., it contains information required to reach other peers;
- `CIndexed`: it stores the portion of the indexing system (globally provided by the full DHT) for which the client is responsible;
- `CKademliaUDPListener`: it manages the operation for incoming network packets used in the management of the DHT.

The search of an object in the overlay is the most important feature offered by Kad, as it is performed each time a user searches for a resource or she has to publish one in the network.

A search in a DHT has always a target which is the identifier of the resource searched. The process contacts peers close to this target, then the request is routed to closer peers, until it finds the target or the closest peer.

All searches executed by a client in the DHT are managed by the class `CSearchManager`. This class provides methods to create the searches and manage their life cycle. Each search is implemented with an instance of the class `CSearch`, which contains all the structures used to store the information about the search and the used contacts.

There are different types of searches, which differs for the type of the target and for the operations done on the retrieved contacts, but all of them share the procedure used to retrieve the peers closer to the target.

The *keyword search* allows the application to search for a shared file in the Kad network. When the user searches for a file name, the target of the keyword search is the hash of the first word in the string. This type of search performs a lookup for the requested target. Then, it sends a request for the keyword with the whole string asked by the user to the peers closest to the target which have responded. Among these peers, some will be responsible for the requested keyword, and they will answer with a list of known shared files that matches the search criterion. The search continues until the searching client has retrieved at least 300 results, the timeout of 45 seconds expires, or there are no more peers to query.

Each result contains some details about the file, such as its full name, the type, the size and an approximate number of available resource providers. Moreover it contains the file hash, needed to univocally identify the file among all the shared ones. When the peer decides to download a file, this hash is used to find all the available resource providers with another type of search, the *provider search*. The target of this type of search is the file hash, which identifies the file and gives a fixed location in the Id space of the DHT. The provider search starts as usual with the lookup for the peers closer to the target, i.e., with an identifier close to the identifier of the file. The closest peers that respond are then queried for the providers of the requested file. Among them, some will be the responsible for the requested file, and they will answer with a list of providers. The search stops for the same reasons of a keyword search: the client has retrieved 300 providers, the timeout expires, or there are no more peers to query. The peer can directly contact the retrieved providers to download the file. This is managed by the eDonkey protocol.

When a peer connects to the Kad network, it can start storing the information about its shared files. This procedure is usually referred to as resource publishing. This process consists of two different types of searches: one to store the information about the resource related keywords, another to add the peer to the list of providers for a file. In essence, resource publishing is managed by searches as the client has to find the responsible peers for the resources it has to share, among all the clients connected to the DHT. Resource publishing of all files and keywords is started and controlled by the class `CSharedFileList`, which starts the procedure whenever needed.

B. CLOSER functionalities

The main goal of this work is to create a CLOSER-aware P2P client. Hence, following the guidelines presented in Section III-A, the most important component to implement is the local DHT, with all the procedures to manage and retrieve the locality information. Furthermore, all the local searches must be included in this new component: keyword searches, provider searches, and the ones that implement resource publishing.

1) The local DHT: We base the local DHT (referred to as *eKad network* in the following) on the implementation of Kademlia provided by aMule. In particular, all the classes needed for the management of the local DHT are replicated and modified to exploit the locality-awareness. The main tasks of the eKad network are managed by the class `CeKademlia`, which operates in the same way as `CKademlia`, the class managing the legacy network. Also other classes of Kad are replicated:

- `CePrefs`: contains all the information about the peer for the local DHT;
- `CeRoutingZone`: stores and manages the local routing table;
- `CeIndexed`: stores the client's share of the local indexing system;
- `CeKademliaUDPListener`: manages the operation for incoming network packets used in the management of the local DHT.

Even if the DHTs are two uncoupled indexing systems, their life cycles are closely related. The eKad network checks the status of the legacy DHT before starting: if Kad is running, the procedure continues, otherwise it is stopped. In this way, when the local DHT is running, also the legacy one is active. This relationship between the two networks is required to ensure correct operation of the application. Indeed, even if most of their operations are totally independent, the eKad depends on Kad for some essential basic operations.

An important functionality highlighting this strict dependence is the management of incoming UDP packets. All UDP socket operations are managed by the class `CClientUDPSocket`, which sends and receives all the packets exchanged among clients. This class parses each packet and reads the first byte to determine the application protocol: if the packet contains the Kad protocol identifier, it is forwarded to the class `CKademlia` to be processed. Since eKad shares the protocol identifier with the legacy counterpart Kad, it is necessary to use the operation code of the Kad protocol to separate eKad packets from the others. This code is the second byte of each Kad UDP packet and is used by the class `CKademliaUDPListener` to correctly process each packet. The local DHT defines its own set of operation codes. Hence, when an eKad packet is received, Kad does not identify the operation. Instead of reporting an error, Kad forwards the packet to the class `CeKademlia`, which uses its instance of the class `CeKademliaUDPListener` to process the packet.

The class diagram in Figure 4 describes the main classes used for the management of the local DHT and the relations among them and with the aMule legacy network Kad.

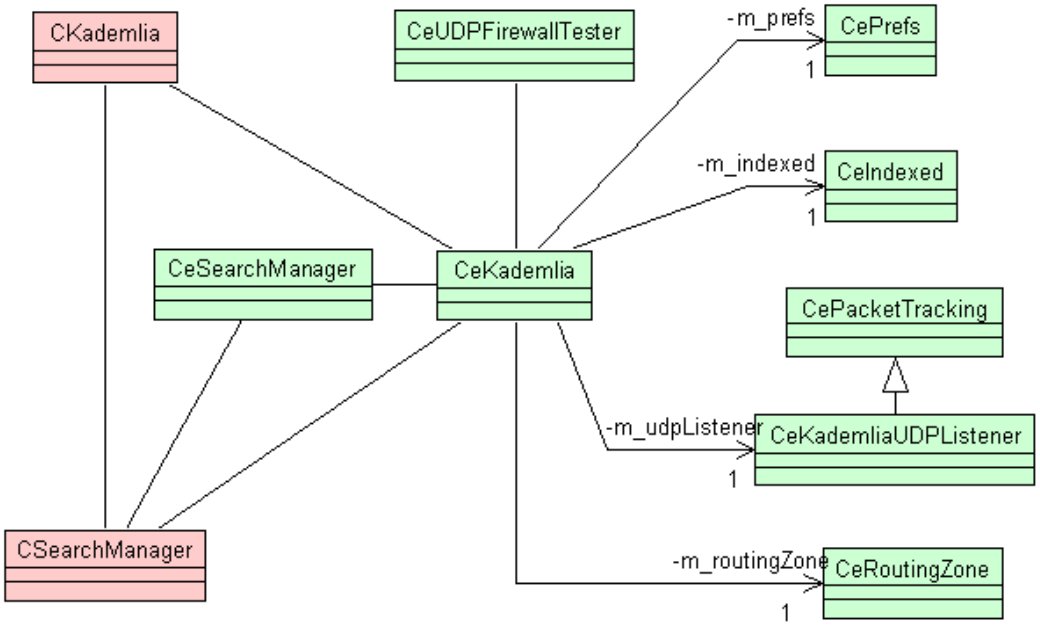

Fig. 4. eKad classes.

2) Localization information retrieval and storage: As described in Section III-A, CLOSER uses three identifiers that, joined, define the localization information: the AS number, the POP ID and the Area ID. These identifiers are managed and stored by the instance of the class `CePrefs`, which already contains the information about the client for the local DHT. Each identifier is stored in a dedicated attribute of the class: a 32 bit long unsigned integer for the AS number, and two 16 bit long unsigned integers for the Area and POP identifiers.

This information is retrieved by the client through a DNS query. The procedure is performed by the method `RetrieveEId(char *name)` of the class `CePrefs`, which is called in the eKad initialization. The method gets the fake name "localization.ep2p" as parameter, which is put in the DNS query for resolution. The query returns two results, which in the fields dedicated to the IP addresses contain the locality information for the client. The first fake IP address returned is actually the AS number (32 bit), while the second is divided in two components: the first 16 bits are used as Area ID, the last 16 as POP ID.

If the client operates in a non collaborative ISP, or the DNS query fails for any other reason, the application falls back to a different less precise approach: it tries to determine the AS number from the local IP address by querying the MaxMind GeoIP database [16], which allows obtaining some localization data (e.g., the country, the city, the ISP, etc.) from an IP address.

The retrieval of the locality information is an essential procedure in the initialization of the eKad network. If both the DNS query and the GeoIP database query fail, the local DHT is not started and CLOSER functionalities cannot be used.

3) The search process: Local keyword and provider searches are closely related to the ones made in the legacy DHT, because they cooperate to retrieve a high number of results. These searches are managed by the class `CeSearchManager`, which controls the life cycle of each single search implemented by the class `CeSearch`.

When a user starts a search for a shared file, the application creates a keyword search through the static method `CSearchManager::Pre` which creates, starts and returns a legacy search object. If the local DHT is enabled and running, the new search is not immediately

started and another search object is created with the class `CeSearchManager`. This object is an instance of the class `CeSearch`. Hence, the search is executed in the local DHT. When the local search ends, the application checks the number of results it returned: if it is greater or equal than the number of results requested to stop a keyword search (300 in the current implementation of Kad), the search process can be stopped; otherwise, the legacy search is started and the results of the two searches are merged.

Once keyword searches are finished and the user starts a download, the application creates a provider search in order to find peers offering the required resource. This is done through the static method `CSearchManager::PrepareLookup()`, which creates a legacy provider search object and, if requested, it starts the search. If the local DHT is enabled and running, the method actually creates a local provider search object with the class `CeSearchManager`. Once the lookup procedure has found the node responsible for the hash value representing the requested resource, the client sends it a request for providers, also including its localization information. The contacted peer uses this information to answer with a list of providers, ordered by increasing distance from the location of the requester. In essence, it inserts firstly providers belonging to the same POP of the requester, then providers belonging to the same Area but different POP, and finally providers belonging to different POP and Area. AS number is implicit as we are dealing with a search in the local DHT. However, also for this type of search, the procedure may continue in the legacy DHT if the number of obtained results is lower than a given value. In this case, the requester also obtains providers belonging to different ISPs.

4) Resource publishing: Publishing of keywords on eKad does not require modifications with respect to its counterpart in the legacy DHT. Instead, publishing the list of providers on the local DHT requires some modifications to actually implement CLOSER. The procedure is performed by the static method `CeKademlia::Publish()`, which in essence adds the localization information of the publisher to the packet used for the publishing request. This packet is sent to the peer responsible for the resource in the local DHT. In this way, the peer can store the localization information of each provider in its local database and then use this information to reply to a provider search with a sorted list of sources, as described above.

C. Experimental results

The utilization of this application for validation purposes is not possible for scalability reasons. In fact, it is impossible to involve and remotely handle a large number of nodes located worldwide, as well as it is not trivial to generate real P2P traffic over these nodes. Instead, this experimental work aims at proving the real feasibility of CLOSER, showing how the locality-aware features can be seamlessly introduced in an existing client. An extensive validation of CLOSER is obtained in [6] by analysis and simulation.

Despite these considerations, some experiments are run over the PlanetLab infrastructure for the sake of completeness. We arbitrarily defined a logical topology over the PlanetLab network consisting of two ASs, each divided in two Areas. Each Area is further divided in four PoPs. Since this division is arbitrary and does not represent a real topology, in these experiments we manually configured the localization information on each peer. Table I and Table II show the list of resource providers obtained by our modified client running on the PlanetLab's machine "planetlab1lannion.elibel.fr" with and without CLOSER, respectively. The node is located in AS 1, Area 1, PoP 3. In the tables, we introduce the notation (*x.y.z*) for the localization information, where *x* represents the AS Number, *y* the Area ID, and *z* the PoP ID. Notice how the list is perfectly ordered when CLOSER is used (i.e., when eKad is enabled). This experiment verifies the proper operation of our CLOSER implementation.

TABLE I. LIST OF RESOURCE PROVIDERS OBTAINED BY THE NODE PLANETLAB1LANNION.ELIBEL.FR (LOCATED IN AS 1, AREA 1, POP 3) WHEN EKAD IS DISABLED

Provider IP address	Location	Search type
160.78.253.32	1.2.3	Kad
192.33.90.66	1.1.2	Kad
195.37.16.97	1.2.4	Kad
87.84.153.115	2.5.3	Kad
155.246.12.163	1.2.1	Kad
144.206.66.58	1.3.1	Kad
87.84.153.114	2.4.4	Kad
137.189.98.32	2.5.4	Kad
132.252.152.193	2.5.4	Kad
193.174.155.35	2.5.2	Kad
128.135.11.152	1.1.4	Kad

V. Conclusion

This paper presents possible design guidelines to implement the Collaborative Locality-aware Overlay SERvice (CLOSER) technology (object of two recent patent applications) in a real P2P system, both considering the traffic locality module and the distributed caching infrastructure that is also part of the architecture. In particular, the paper shows how CLOSER can be introduced in a totally distributed DHT-based P2P system through the development of additional DHTs within the boundaries of the various Autonomous Systems. Searches are first done in the local DHT and possibly continued in the traditional global DHT if the number of retrieved results is not sufficient.

The paper also presents a heuristic to drive content selection in the proactive caching system also provided within CLOSER. Simulation results show the potential of the approach when bandwidth throttling is applied on backbone links.

Finally, the paper describes a real CLOSER implementation based on the aMule P2P client. Main modifications required are discussed and analyzed. Although the main aim of this experimental work is to prove the real feasibility of CLOSER, some experiments were run on the PlanetLab infrastructure and demonstrated the correct operation of the application.

References

[1] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in BitTorrent via biased neighbor selection," in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, 2006.

[2] D. R. Choffnes and F. E. Bustamante, "Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. Seattle, WA, USA: ACM, 2008, pp. 363–374.

[3] "Kontiki." [Online]. Available: http://www.kontiki.com

[4] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPS and P2P users cooperate for improved performance?" *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, pp. 29–40, 2007.

[5] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. G. Liu, and A. Silberschatz, "P4P: provider portal for applications," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. Seattle, WA, USA: ACM, 2008, pp. 351–362.

[6] M. Papa Manzillo, L. Ciminiera, G. Marchetto, and F. Risso, "Closer: A collaborative locality-aware overlay service," *Parallel and Distributed Systems, IEEE Transactions on*, to appear.

[7] L. Ciminiera, M. Papa Manzillo, V. Vercellone, and M. Ullio, "Sharing of digital content in p2p networks exploiting localization data," Patent Application PCT/EP 064 546, 2009.

[8] ——, "Improved caching of digital contents in p2p networks," Patent Application PCT/EP 064 547, 2009.

[9] "aMule." [Online]. Available: http://www.amule.org/

[10] J. Seedorf, S. Kiesel, and M. Stiemerling, "Traffic localization for p2p-applications: The alto approach," in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, sep. 2009, pp. 171 –177.

[11] M. J. Freedman and R. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002, pp. 193–206.

[12] M. Rennhard and B. Plattner, "Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection," in *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. New York, NY, USA: ACM, 2002, pp. 91–102.

TABLE II. List of resource providers obtained by the node planetlab1lannion.elibel.fr (located in AS 1, Area 1, PoP 3) when eKad is enabled

Provider IP address	Location	Search type
192.43.193.71	1.1.3	eKad
130.192.157.132	1.1.2	eKad
128.135.11.152	1.1.4	eKad
155.246.12.164	1.1.1	eKad
128.195.54.161	1.2.1	eKad
130.92.70.252	1.2.1	eKad
144.206.66.58	1.3.1	eKad
141.24.33.192	1.3.1	eKad
213.131.1.102	2.4.2	Kad
193.174.155.35	2.5.2	Kad
87.84.153.115	2.5.3	Kad
144.206.66.56	2.4.3	Kad
198.175.112.105	2.5.4	Kad

[13] V. Aggarwal, O. Akonjang, A. Feldmann, R. Tashev, and S. Mohr, "Reflecting P2P user behaviour models in a simulation environment," in *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, 2008, pp. 516–523.

[14] O. Saleh and M. Hefeeda, "Modeling and caching of Peer-to-Peer traffic," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, 2006, pp. 249–258.

[15] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer information system based on the XOR metric," in *Peer-to-Peer Systems*, 2002, pp. 53–65.

[16] MaxMind LLC, "GeoIP." [Online]. Available: http://www.maxmind.com/