Content Distribution in P2P Systems

Manal El Dick, Esther Pacitti

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Content Distribution in P2P Systems

Manal El Dick — Esther Pacitti

N° 7138

Décembre 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--7138--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Content Distribution in P2P Systems*

Manal El Dick †, Esther Pacitti‡

Thème COM — Systèmes communicants
Équipe-Projet Atlas

Rapport de recherche n° 7138 — Décembre 2009 — 63 pages

Abstract: The report provides a literature review of the state-of-the-art for content distribution. The report's contributions are of threefold. First, it gives more insight into traditional *Content Distribution Networks* (CDN), their requirements and open issues. Second, it discusses *Peer-to-Peer* (P2P) systems as a cheap and scalable alternative for CDN and extracts their design challenges. Finally, it evaluates the existing P2P systems dedicated for content distribution according to the identified requirements and challenges.

Key-words: CDN, P2P, content distribution

* Work partially funded by the DataRing project of the french ANR.
† INRIA et LINA, Université de Nantes
‡ INRIA et LIRMM, Université de Montpellier 2

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Distribution de contenu dans les systèmes P2P

Résumé : Dans ce rapport, nous dressons un état de l'art des systèmes de distribution de contenu. Nous apportons trois contributions principales. En premier lieu, nous donnons un aperçu global sur les réseaux de distribution de contenu, leurs exigences et leurs problèmes. En second lieu, nous étudions les systèmes P2P en tant qu'alternative efficace aux CDNs et nous identifions leurs défis. Enfin, nous évaluons les systèmes P2P existants qui sont conçus pour la distribution de contenu.

Mots-clés : CDN, P2P, distribution de contenu

1 Introduction

The explosive growth of the Internet has triggered the conception of massive scale applications involving large numbers of users in the order of thousands or millions. According to recent statistics [37], the world had 1.5 billion Internet users by the end of 2007. The client-server model is often not adequate for applications of such scale given its centralized aspect. Under this model, a content provider typically refers to a centralized web-server that exclusively serves its content (e.g., web-pages) to interested clients. Eventually, the web-server suffers congestion and bottleneck due to the increasing demands on its content [96]. This substantially decreases the service quality provided by the web-server. In other terms, the web-server gets overwhelmed with traffic due to a sudden spike in its content popularity. As a result, the website becomes temporarily unavailable or its clients experience high delays mainly due to long download times, which leaves them in frustration. That is why the World Wide Web is often pejoratively called World Wide Wait [58].

In order to improve the Internet service quality, a new technology has emerged that efficiently delivers the web content to large audiences. It is called *Content Distribution Network* or *Content Delivery Network* (CDN) [5]. A commercial CDN like Akamai[1] is a network of dedicated servers that are strategically spread across the Internet and that cooperate to deliver content to end-users. A content provider like Google and CNN can sign up with a CDN so that its content is deployed over the servers of the CDN. Then, the requests for the deployed content are transparently redirected to and handled by the CDN on behalf of the origin web-servers. As a result, CDNs decrease the workload on the web-servers, reduce bandwidth costs, and keep the user-perceived latency low. In short, CDNs strike a balance between the costs incurred on content providers and the QoS provided to the users [63]. CDNs have became a huge market for generating large revenues [36] since they provide content providers with the highly required *scalabiliy*, *reliability* and *performance*. However, CDN services are quite expensive, often out of reach for small enterprises or non-profit organizations.

The new web trend, Web 2.0, has brought greater collaboration among Internet users and encouraged them to actively contribute to the Web. *Peer-to-Peer* (P2P) networking is one of the fundamental underlying technologies of the new world of Web 2.0. In a P2P system, each node, called a *peer*, is client and server at the same time – using the resources of other peers, and offering other peers its own resources. As such, the P2P model is designed to achieve *self-scalability* : as more peers join the system, they contribute to the aggregate resources of the P2P network. P2P systems that deal with content sharing (e.g., sharing files or web documents) can be seen as a form of CDN, where peers share content and deliver it on each other's behalf [80]. The more popular the content (e.g., file or web-page), the more available it becomes as more peers download it and eventually provide it for others. Thus, the P2P model stands in direct contrast to traditional CDNs like Akamai when handling increasing amounts of users and demands. Whereas a CDN must invest more in its infrastructure by adding servers, new users bring their own resources into a P2P system. This implies that P2P systems are a perfect match for building cheap and scalable CDN

[1] `http://www.akamai.com`

infrastructures. However, making use of P2P self-scalability is not a straightforward endeavor because designing an efficient P2P system is very challenging.

This report reviews the state-of-the-art for both traditional and P2P content distribution in order to identify the shortcomings and highlight the challenges.

Roadmap. The rest of the report is organized as follows. Section2 gives more insight into traditional CDNs and highlights their requirements which are needed for the design of novel and cheaper alternatives. Section 3 presents P2P systems and identifies their fundamental design requirements. Section 4 investigates the recent P2P trends that are useful for content distribution and identifies their challenges. Then, Section 5 deeply explores the state-of-art in P2P solutions for content distribution. It evaluates the existing approaches against the previously identifed requirements (for both P2P and CDN) and enlightens open issues.

2 Insights on Content Distribution Netwoks

Content distribution networks is an important web caching application. First, let us briefly review the different web caching techniques in order to position and understand the CDN technology. Then, we shed lights on CDNs, their requirements and their open issues.

2.1 Background on Web Caching

A web cache is a disk storage of predefined size that is reserved for content requested from the Internet (such as HTML pages and images)[2]. After an original request for an object has been successfully fulfilled, and that object has been stored in the cache, further requests for this object results in returning it from the cache rather than the original location. The cache content is temporary as the objects are dynamically cached and discarded according to predefined policies (further details in Section 2.2.1).

Web caching is widely acknowledged as providing three major advantages [6]. First, it reduces the bandwidth consumption since fewer requests and responses need to go over the network. Second, it reduces the load on the web-server which handles fewer requests. Third, it reduces the user-perceived latency since a cached request is satisfied from the web cache (which is closer to the client) instead of the origin web-server. Together, these advantages make the web less expensive and better performing.

Web caching can be implemented at various locations using *proxy servers* [96, 58]. A *proxy server* acts as an intermediary for requests from clients to web-servers. It is commonly used to cache web-pages from other web-servers and thus intercepts requests to see if it can fulfill them itself. A proxy server can be placed in the user's local computer as part of its web browser or at various points between the user and the web-servers. Commonly, proxy caching refers to the latter schemes that involve dedicated servers out on the network while the user's local proxy cache is rather known as *browser cache*.

[2]Web caching is different from traditional caching in main memory that aims at limiting disk accesses

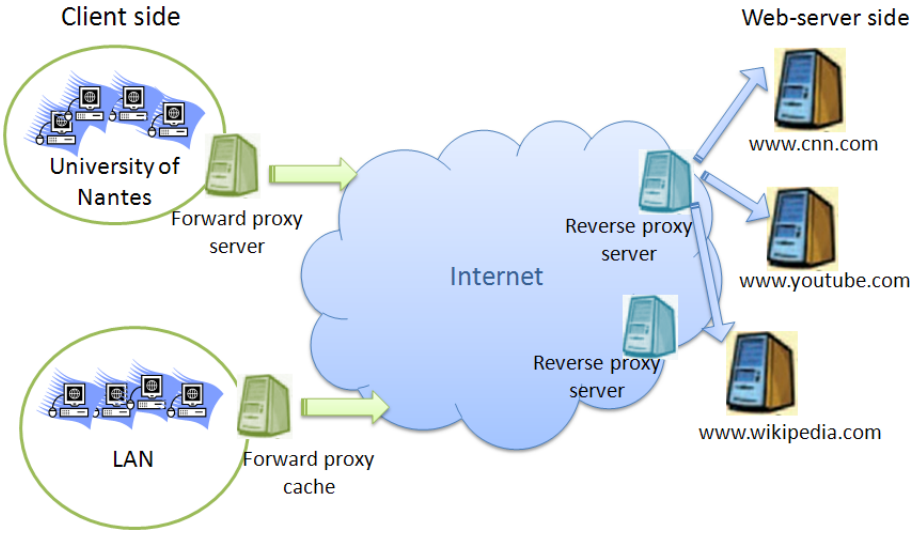


Figure 1: Web caching: different placements of proxy servers.

Depending on their placement and their usage purpose, we distinguish two kinds of proxies, *forward proxies* and *reverse proxies*. They are illustrated in Figure 1.

A *forward proxy* is used as a gateway between an organisation (i.e., a group of clients) and the Internet. It makes requests on behalf of the clients of the organisation. Then, it caches requested objects to serve subsequent requests coming from other clients of the organisation. Large corporations and Internet Service Providers (ISP) often set up forward proxies on their firewalls to reduce their bandwidth costs by filtering out repeated requests. As illustrated in Figure 1, the university of Nantes has deployed a forward proxy that interacts with the Internet on behalf of the university users and handles their queries.

A *reverse proxy* is used in a network in front of web-servers. It is delegated the authority to operate on behalf of these web-servers, while working in close cooperation with them. Typically, all requests addressed to one of the web-servers are routed through the proxy server which tries to serve them via caching. Figure 1 shows a reverse proxy that acts on behalf of the web-servers of wikipedia.com, cnn.com and youtube.com by handling their received queries. A CDN deploys reverse proxies throughout the Internet and sells caching to websites that aim for larger audience and lower workload. The reverse proxies of a CDN are commonly known as *surrogate survers*.

2.2 Overview of CDN

A CDN deploys hundreds of surrogate servers around the globe, according to complex algorithms that take into account the workload pattern and the network topology [65]. Figure 2 gives an overview of a CDN that distributes and delivers the content of a web-server in the US.

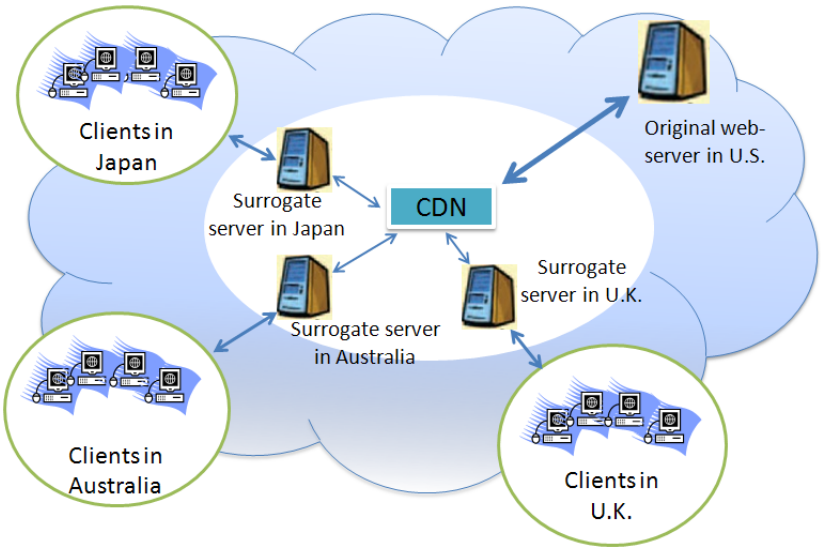


Figure 2: Overview of a CDN.

Examples of commercial CDNs are Akamai [3] and Digital Island [4]. They mainly focus on distributing static content (e.g., static HTML pages, images, documents, audio and video files), dynamic content (e.g., HTML or XML pages that are generated on the fly based on user specification) and streaming audio or video. Further, ongoing research aims at extending CDN technology to support video on demand (VoD) and live streaming. In this paper, we mainly focus on static content. This type of content has a low frequency of change and can be easily cached; its freshness can be maintained via traditional caching policies [96].

A CDN stores the content of different web-servers and therefore handles related queries on behalf of these web-servers. Each website selects specific or popular content and pushes it to the CDN. Clients requesting this content are then redirected to their closest surrogate server via *DNS redirection* or *URL rewriting*. The CDN manages the replication and/or caching of the content among its surrogate servers. These techniques are explained in more detail below.

The interaction between a user and a CDN takes place in a transparent manner, as if it is done with the intended origin web-server. Let us consider a typical user interaction with the well-known CDN, Akamai [91], which mainly deals with objects embedded in a web-page (e.g., images, scripts, audio and video files). First, the user's browser sends a request for a web-page to the website. In response, the website returns the appropriate HTML page as usual, the only difference being that the URLs of the embedded objects in the page have been modified to point to the Akamai network. As a result, the browser

[3] `http://www.akamai.com`
[4] `http://www.digitalisland.com/`

next requests and obtains the embedded objects from an optimal surrogate server.

How is this transparent interaction achieved from a technical perspective? In the following, we investigate this issue by exploring CDN techniques for replication and caching on one hand, and location and routing on the other hand.

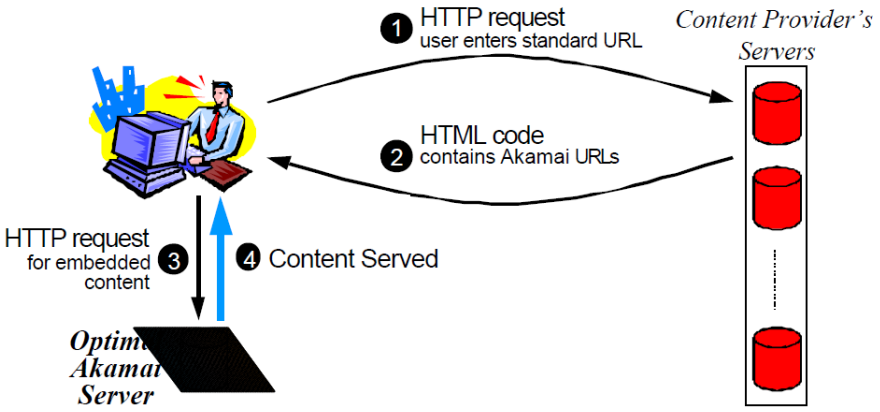


Figure 3: Typical user interaction with a website using Akamai services [91].

2.2.1 Replication and Caching in CDN

According to [14] and [87], replication involves creating and permanently maintaining duplicate copies of content on different nodes to improve content availability. One the other hand, caching consists in temporarily storing passing by request responses (e.g., web-pages, embedded objects like images) in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Note that web documents are typically accessed in read-only mode: requests read a document without changing its contents.

Replication. In a CDN, replication is typically initiated when the origin web-servers pushes content to any surrogate servers [63, 5]. The surrogate servers then manage the replication of the content among each other, either on-demand or beforehand.

In on-demand replication, the surrogate server that has received a query and experienced a cache miss, pulls the requested content from the origin web-server or other surrogate servers. In the latter case, it might use a centralized or distributed index to find a nearby copy of the requested content within the CDN.

Beforehand replication implies different strategies that replicate objects a priori and dynamically adjust their placement in a way that brings them closer to the clients and balances the load among surrogate servers [65].

However, due to replication requirements in terms of cost and time, any replicas' placement should be static for a large amount of time.

Caching. Given that popularity of objects may fluctuate with time, some replicas may become redundant and unnecessary. This leads to unoptimized storage management at surrogate servers. That is why caching can be seen as an interesting alternative to replication, especially in cases where unpredictable numerous users have suddenly interest in the same content.

Content objects are dynamically cached and evicted from the cache according to cache replacement policies. More precisely, each cached object is assigned a cache expiration policy which defines how long it is fresh based on its own characteristics [96]. Upon receiving a request for an object, the server first checks the freshness of its cached version before serving it. In case it has expired, the surrogate server checks with the origin server if the object has changed (by sending *a conditional GET (cGET)* request, e.g. *If-Modified-Since* request). Subsequently, the origin server either validates the cached copy or sends back a fresh copy. Since the cache has a limited storage size, the server might need to evict cached objects via one of the cache replacement policies that have been studied in [96]. In the policy LRU, the rarely requested objects stored in the local cache are replaced with the new incoming objects. Additionnaly, the cache may regularly check for expired objects and evict them.

An evaluation of caching and replication as seperate approaches in CDNs is covered in [46], where caching outperforms but replication is still preferred for content availability and reliability of service. If replication and caching cooperate they may greatly fortify the CDN since both deal with the same problem but from a different approach. Indeed, [87] has proved that potential performance improvement is possible in terms of response time and hit ratio if both techniques are used together in a CDN. CDNs may take advantage of the dynamic nature of cache replacement policies while maintaining static replicas for availability and reliability.

2.2.2 Location and Routing in CDN

To serve a query in a CDN, there are two main steps, server location and query routing. The first step defines how to select and locate an appropriate surrogate server holding a replica of the requested object whereas the second step consists in routing the query to the selected surrogate server. In several existing CDNs, these two steps are combined together in a single operation.

A query routing system uses a set of metrics in selecting the surrogate server that can best serve the query. The most common metrics include proximity to the client, bandwidth availability, surrogate load and availability of content. For instance, the distance between the client and a surrogate server can be measured in terms of *round-trip-time*(RTT) via the common tool of "ping".

Actually, each CDN uses its proprietary algorithms and mechanisms for location and routing and does not always reveal all the technology details. Here, we try to give a generic description of the mechanisms commonly used by CDNs, based on the materials in [5, 65]. The most common query routing technique are *DNS redirection* and *URL rewriting*.

DNS Redirection. CDNs can perform dynamic request routing using the Internet's *Domain Name System* (DNS). The DNS is a distributed directory service for the Internet whose primary role is to map *fully qualified domain*

names (FQDNs) to IP addresses. For instance, hostname.example.com translates to 208.77.188.166. The DNS distributes the responsibility of assigning domain names and mapping those names to IP addresses over *authoritative name servers*: an *authoritative name server* is designated to be responsible for each particular domain, and in turn can designate other authoritative name servers for its sub-domains. This results in a hierarchical authority structure that manages the DNS. To determine an FQDN's address, a DNS client sends a request to its local DNS server which resolves it by recursively querying a set of authoritative DNS servers. When the local DNS server receives an answer to its request, it sends the result to the DNS client and caches it for future queries.

Normally, DNS mapping from an FQDNs to an IP address is static. However, CDNs use modified authoritative DNS servers to dynamically map each FQDN to multiple IP addresses of surrogate servers. The query answer may vary depending on factors such as the locality of the client and the load on the surrogate servers. Typically, the DNS server returns, for a request, several IP addresses of surrogate servers holding replicas of the requested object. The DNS client chooses a server among these. To decide, it may issue pings to the servers and choose based on resulting RTTs. It may also collect historical information from the clients based on previous access to these servers.

URL Rewriting. In this approach, the origin web-server redirects the clients to different surrogate servers by rewriting the URL links in a web-page. For example, with a web-page containing an HTML file and some embedded objects, the web-server would modify references to embedded objects so that they point to the CDN or more particularly to the best surrogate server. Thus, the base HTML page is retrieved from the origin web-server, while embedded objects are retrieved from CDN servers. To automate the rewriting process, CDNs provide special scripts that transparently parse web-page content and replace embedded URLs. URL rewriting can be pro-active or reactive. In the pro-active URL rewriting, the URLs for embedded objects of the main HTML page are formulated before the content is loaded in the origin server. In reactive approach involves rewriting the embedded URLs of a HTML page when the client request reaches the origin server.

2.3 Requirements and Open Issues of CDN

As introduced previously, a CDN has to fulfill stringent requirements which are mainly *reliability*, *performance* and *scalability* [63].

- **Reliability** guarantees that a client can always find and access its desirable content. For this, the network should be robust and avoid single point of failure.
- **Performance** mainly involves the response time perceived by end-users submitting queries. Slow response time is the single greatest contributor to clients abandoning web-sites [92].
- **Scalability** refers to the adaptability of the network to handle more amounts of content, users and requests without significant decline in performance. For this, the network should prevent bottlenecks due to overload situations.

The reliability and performance of a CDN is highly affected by the mechanisms of content distribution as well as content location and routing. Content distribution defines how the content is distributed over the CDN and made available for clients. It mainly deals with the placement of content and involves caching and replication techniques in order to make the same content accessible from several locations. Thus, with these techniques, the content is located near to the clients yielding low response times and high content availability since many replicas are distributed. Content location and routing defines how to locate the requested content and route requests towards the appropriate and relevant servers. *Locality-awareness* refers to any topological information about the localities of peers to be able to evaluate their physical proximity. *Locality-awareness* is a top priority in routing mechanisms in order to find content close to the client in locality.

To expand and scale-up, CDNs need to invest siginficant time and costs in provisioning additional infrastructures [92]. Otherwise, they would compromise the quality of service received by individual clients. Further, they should dynamically adapt their resource provisioning in order to address unexpected and varying workloads. This inevitably leads to more expensive services for websites. In the near future, the clients will also have to pay to receive high quality content (in some of today's websites like CNN.com, users have already started to pay a subscription to view videos). In this context, scalability will be an issue to deliver high quality content, maintaining low operational costs [5].

Most recently, traditional CDNs [5] have turned towards P2P technology to reduce investments in their own infrastructure, in the context of video streaming. The key idea is to dynamically couple traditional CDN distribution with P2P distribution. Basically, the CDN serves a handful of clients which in turn provide the content to other clients. Joost[5] and BitTorrent[6] are today's most representative CDN companies using P2P technology to deliver Internet television and video streaming, respectively.

To conclude this section, we observe that P2P technology is being progressively accepted and adopted as a mean of content distribution. The existing CDNs still depend –at least partly- on a dedicated infrastructure, which requires investment and centralized administration. If the CDN could rely on a cheap P2P infrastructure supported only by end-users, this would provide a cheap and scalable alternative that avoids the considerable costs. In the following, we further investigate the feasibility of pure P2P content distribution.

3 P2P Systems

In the past few years, P2P systems have emerged as a popular way to share content. The most representative systems include Gnutella [30, 42, 41], BitTorrent[7] [66] and Fastrack/Kazaa [51]. The popularity of P2P systems is attested by the

[5] `http://www.joost.com`

[6] The technology is called *BitTorrent DNA* (*Delivery Network Accelerator*). Available at `http://www.bittorrent.com/dna/`

[7] `http://www.bittorrent.com/`

fact that the P2P traffic accounts for more than 70% of the overall Internet traffic according to a recent study[8].

The P2P model holds great promise for decentralized collaboration among widespread communities with common interests. This communal collaboration lies at the heart of the Web 2.0 paradigm and stems from the principle of resource sharing. By distributing storage, bandwidth and processing across all participating peers, P2P systems can achieve high scalability, that would otherwise depend on expensive and dedicated infrastructure.

Let us first give an overview of P2P systems by defining their main concepts then we can explore them in more detail.

3.1 Overview of P2P Systems

P2P systems operate on application-level networks referred to as *overlay networks* or more simply *overlays*. In other words, peers are connected via a logical overlay network superposed on the existing Internet infrastructure. When two peers are connected via a logical link, this implies that they know each other and can regularly share information across this link. We say that the peers are *neighbors* in the P2P network. Figure 4 shows a P2P overlay network where Peer A and Peer B are neighbors, independently of their Internet location. A

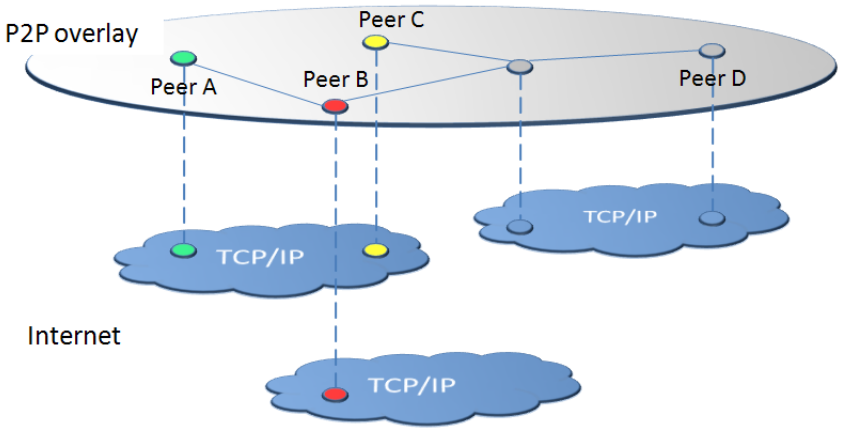


Figure 4: P2P overlay on top of the Internet infrastructure.

P2P overlay network serves as an infrastructure for applications that wish to exploit P2P features. It relies on a *topology* and its associated *routing protocol*. The overlay topology defines how the peers are connected whereas the routing protocol defines how the messages are routed between peers. According to their degree of structure, P2P overlays can be classified into two main categories: *structured* and *unstructured*. Typically, they differ on the constraints imposed on how peers are organized and where shared objects are placed [67].

[8]Available at `http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009`.

The P2P overlay has a direct impact on the performance, reliability and scalability of the system. Given that P2P networks operate in open and vulnerable environments, peers are continuously connecting and disconnecting, sometimes unexpectedly failing. The arrival and departure of peers by thousands creates the effect of churn [90] and requires a constant restructuring of the network core. For the purpose of reliability, the P2P overlay must be designed in a way that treats failures and churn as normal occurences. For the purpose of scalability, the P2P overlay should dynamically accommodate to growing numbers of participants. The performance of P2P systems refers to their efficiency in locating desirable content, which tightly depends on the P2P overlay, mainly on the routing protocol.

Basically, when a peer searches for a given object, it originates a query and routes it over the P2P overlay. Whenever a peer receives the query, it searches its local repository for the requested object. Eventually, the query reaches a peer that can satisfy the query and respond to the requester. The responder peer is either able to provide a copy of the requested object or has a pointer to the location of the object. Accordingly, the responder peer generates a *query response* that contains along with the object information (e.g., filename, id), the address of the provider peer. Upon receiving the query response, the query originator downloads a copy of the object from the provider peer.

In the following, we present the two categories of P2P overlays, i.e., unstructured and structured overlays. For each category, we discuss its behavior under churn as well as its strengths and weaknesses. Then, we summarize by stating the requirements of P2P systems and accordingly compare both categories.

3.2 Unstructured Overlays

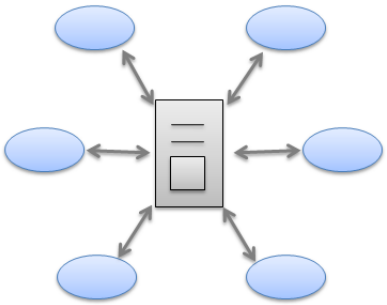
Often referred to as the first generation P2P systems, unstructured overlays remain highly popular and widely deployed in today's Internet. They impose loose constraints on peer neighborhood (i.e, peers are connected in an ad-hoc manner) and content placement (i.e., peers are free to place content anywhere) [67].

3.2.1 Decentralization Degrees

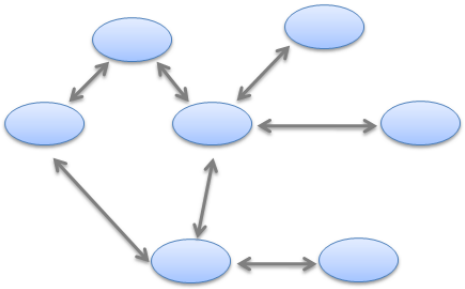
Although P2P systems are supposed to operate in a fully decentralized manner, in practice, we observe that various degrees of decentralization can be applied to the routing protocols of unstructured overlays. Accordingly, we classify unstructured overlays into three groups: *centralized*, *pure decentralized* and *partially decentralized with superpeers*.

Centralized In these overlays, a central server is in charge of indexing all the peers and their shared content as shown in Figure 5a. Whenever a peer requests some content, it directly sends its query to the central server which identifies the peer storing the requested object. Then, the file is transferred between the two peers. The now-defunct Napster[9] [8] adopted such a centralized architecture.

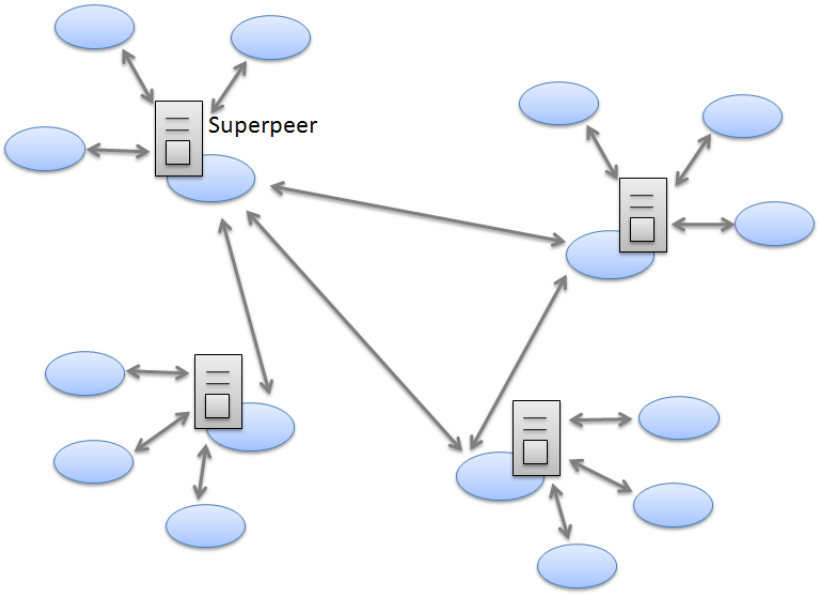
[9] `http://www.napster.com/`



(a) Centralized.



(b) Pure decentralized.



(c) Partially decentralized with superpeers.

Figure 5: Types of unstructured P2P overlays.

Pure Decentralized In pure decentralized overlays, all peers have equal roles as shown in Figure 5b. Each peer can issue queries, serve and forward queries of other peers. Query routing is typically done by blindly flooding the query to neighbors. The flooding mechanism has been further refined, in a way that nowadays we find several variants of flooding like *random walks* and *iterative deepening*. These techniques are explained in more detail in Section 3.2.2. Of the many existing unstructured P2P systems, Gnutella [30, 42, 41] is one of the original pure decentralized networks.

Partially Decentralized with Superpeers In these overlays, high-capacity peers are assigned the role of superpeers, and each superpeer is reponsible of a set of peers, indexing their content and handling queries on their behalf. Superpeers are then organized in a pure decentralized P2P network and can communicate to search for queries (see Figure 5c). They can be dynamically elected and replaced in the presence of failures. Gnutella2 [82] is another version of Gnutella that uses superpeers; Edutella [59] and FastTrack/Kazaa [51] are also popular examples of hybrid networks.

The higher is the degree of decentralization, the more the network is fault-tolerant and robust against failures, because there will be no single point of failure due to the symmetry of roles. However, the higher is the degree of centralization, the more efficient is the search for content. Thus, the hybrid overlay strikes a balance between the efficiency of centralized search, and the load balancing and robustness provided by means of decentralization. Furthermore, it can take advantage of the heterogeneity of capacities (e.g., bandwidth, processing power) across peers. That is why recent generations of unstructured overlays are evolving towards hybrid overlays.

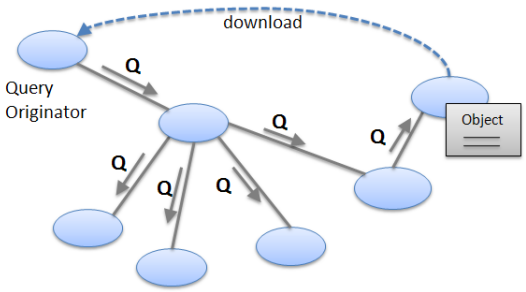
3.2.2 Decentralized Routing Techniques

In decentralized routing, *blind techniques* are commonly used to search for content in unstructured networks. *Blind techniques* route the query without any information related to the location of the requested object. A peer only keeps references to its own content, without maintaining any information about the content stored at other peers. Blind techniques can be grouped into three main categories: *breadth-first-search*, *iterative deepening* and *random walk*.

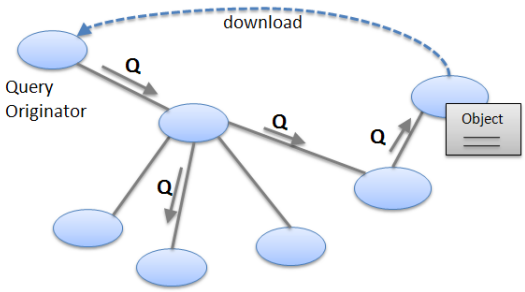
Breadth-First-Search (BFS). Originally, unstructured systems relied on the *flooding* mechanism which is more formally called *Breadth-First-Search* (BFS). Illustrated in Figure6a, the query originator sends its query Q to its neighbors, which in turn forward the message to all their neighbors except the sender and so on. The query is associated with a *Time-To-Live* (TTL) value, which is decreased by one when it travels across one hop in the P2P overlay. At a given peer, the message comes to its end if it becomes redundant (i.e., no further neighbors) or the TTL value becomes zero. Query responses follow the reverse path of the query, back to the requester peer. The main merits of this approach are its simplicity, reliability, and its high network coverage, i.e., a large number of peers could be reached within a small number of hops. However, measurements in [74] have shown that although 95% of any two peers are less than 7 hops away, flooding generates 330 TB/month in a Gnutella network with only 50,000 nodes. This heavy traffic compromises the benefits of unstructured systems and drastically limits their scalability. The reasons behind the traffic burden of flooding are *blindness* and *redundancy*. First, a peer blindly forwards the query without any knowledge about how the other peers can contribute to the query. Second, a peer may receive the same query message multiple times because of the random nature of connections in an unstructured overlay. This can result in huge amounts of redundant and unnecessary messages.

In [43], *modified BFS* has been proposed in attempt to reduce the traffic overhead of flooding. Upon receiving a query, a peer randomly chooses a ratio

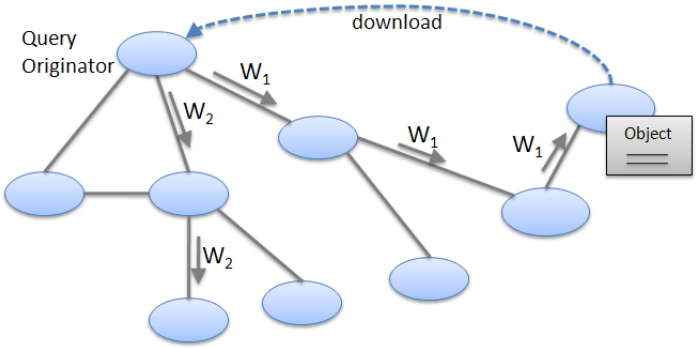
of its neighbors to send or forward the query (see Figure6b). However, this approach may loose many of the good answers which could be found by BFS.



(a) BFS or flooding.



(b) Modified BFS.



(c) Random walk.

Figure 6: Blind routing techniques of unstructured overlays.

Iterative Deepening. This approach [98, 54] is also called *expanding ring*. The query originator performs consecutive BFS searches with successively larger TTL. A new BFS follows the previous one by expanding the TTL, if the query has not been satisfied after a predefined time period. The algorithm ends when the required number of answers is found or the predefined maximum TTL is reached. In case the results are not in the close neighborhood of the query originator, this approach does not address the duplication issue and adds considerable delay to the response time.

Random Walk. In the standard algorithm, the query originator randomly selects one of its neighbors and forwards the query to that neighbor. The latter, in turn, forwards the query to one randomly chosen neighbor, and so on until

the query is satisfied. Compared to the basic BFS, this algorithm reduces the network traffic, but massively increases the search latency.

In the *k-walker random walk* algorithm [54], the query originator forwards k query messages to k randomly chosen neighbors (k is a value specified by the application). Each of these messages follows its own path, having intermediate peers forward it to one randomly chosen neighbor at each step. These query messages are also known as walkers and are shown in (Figure6c) as $W1$ and $W2$. When the TTL of a walker reaches zero, it is discarded. Each walker periodically contacts the query originator, asking whether the query was satisfied or not. If the response is positive, the walker terminates. This algorithm achieves a significant message reduction since it generates, in the worst case, $k * TTL$ routing messages, independently of the underlying network. Nevertheless, a major concern about this algorithm is its highly variable performance because success rates are highly variable and dependable on the network topology and the random choices made. In addition, the random walk technique does not learn anything from its previous successes or failures.

3.2.3 Behavior under Churn and Failures

It is well known that P2P networks are characterized by a high degree of churn [31]. Therefore, it is vital to examine the behavior of P2P networks in highly dynamic environments where peers join and leave frequently and concurrently.

The maintenance of unstructured overlays merely rely on the messages ping, pong and bye: pings are used to discover hosts on the network, pongs are replies to pings and contain in formation about the responding peer and other peers it knows about, and byes are optional messages that inform of the upcoming closing of a connection.

After joining the overlay network (by connecting to boostrap peers found in public databases), a peer sends out a ping message to any peer it is connected to. The peers send back a pong message identifying themselves, and also propagate the ping message to their neighbors. When a peer gets in contact with a new peer, it can add it as a neighbor in its routing table in a straigthforward manner. A peer that detects the failure or leave of a neighbor simply removes it from its routing table. If a peer becomes disconnected by the loss of all of its neighbors, it can merely repeat the bootstrap procedure to re-join the network [10].

The measurements in [67] show that the bandwidth consumption due to maintenance messages is reasonably low in the unstructured Gnutella system. Peers joining and leaving the Gnutella network have little impact on other peers or on the placement of shared objects, and thus do not result in significant maintenance traffic.

To resume, there are few constraints on the overlay construction and content placement in unstructured networks: peers set up overlay connections to an arbitrary set of other peers they know, and shared objects can be placed at any peer in the system. The resulting random overlay topology and content distribution provides high robustness to churn [67]. Furthermore, the routing mechanism greatly rely on flooding which yields randomness and repetitiveness and thus more robustness. Given that a query takes several parallel routes, the disruption of some routes due to peer failures does not prevent the query from being propagated throughout the P2P network.

3.2.4 Strengths and Weaknesses

Unstructured P2P systems exhibit many simple yet attractive features, such as high flexibility and robustness under churn and failures. For instance, the freedom in content placement provides maximum flexibility in selecting policies for replication and caching.

Unstructured overlays are particularly used to support file-sharing applications for two main reasons. First, since they introduce no restritions on the manner to express a query, they are perfectly capable of handling *keyword search*, i.e., searching for files using keywords instead of the exact filenames. Second, file popularity derives a kind of natural file replication among peers, which induces high availability. Indeed, peers replicate the copies of files they request when they download them.

However, the main Achilles heel of unstructured systems are their blind routing mechanisms which incur severe load on the network and give no guarantees on lookup efficiency. Because of the topology randomness, a query search necessitates $O(n)$ hops (where n is the total number of peers), generates many redundant messages and is not guaranteed to find the requested object. Many studies such as [74] and [75] claim that the high volume of search traffic threatens the continued growth of unstructured systems. Indeed, the measurements in [74] have shown that although 95% of any two peers are less than 7 hops away, flooding generates 330 TB/month in a Gnutella network with only 50,000 nodes.

3.3 Structured Overlays

The evolution of research towards structured overlays has been motivated by the poor scaling properties of unstructured overlays. Structured networks discard randomness and impose specific constraints on the overlay topology [67]. They remedy to the blind search by tightly controlling the content placement. As a result, they provide an efficient, deterministic search: they can locate any object within a bounded number of hops.

More precisely, a structured overlay provides a distributed index scheme, by mapping content to locations (e.g., an object identifier is mapped to a peer address). To achieve this, objects and peers are assigned unique identifiers (respectively *keys* and *IDs*) from the same identifier space (e.g., hashing filename or url for an object and the IP address for a peer). Then, this identifier space is dynamically partitionned among peers, so that each peer is responsible for a specific key space partition. Accordingly, a peer stores the objects or pointers related to objects with respect to its key partition. The topology dictates for each peer a certain number of neighbors. The peer holds a routing table that associates its neighbors's identifiers to their IP addresses. Then a routing algorithm is defined to allow a deterministic key-based search. The main representative of structured overlays is the *Distributed Hash Table* (*DHT*) which is presented and dicussed in the following.

3.3.1 DHT Routing

At a fundamental level, DHTs can be viewed as content addressing and lookup engines. A DHT provides *content and peer addressing* via *consistent hashing*

[45]. This technique enables a uniform hashing of values and thereby evenly places or maps content to peers. The addressing mechanism serves as a distributed and semantic-free index, because it gives information about the location of content based on hash-based keys.

The *lookup engine* of the DHT mainly consists in locating the target peer by means of routing over the overlay. The routing protocol tightly depends on the different implementations of DHT and more precisely the *routing geometries* [32]. Nonetheless, all routing protocols aim at providing efficient lookups as well as minimizing the routing state[10] that should be maintained at each peer. Most of them exhibit almost similar space and time complexity. That is, the routing table of peer contains at most $O(log\,N)$ entries and a lookup is normally performed in $O(log\,N)$ hops where N is the total number of nodes in the DHT [34].

The routing geometry mainly defines the manner in which neighbors and routes are established. According to [32], there are 6 basic types of routing geometries: *tree*, *hypercube*, *ring*, *butterfly*, *XOR* and *hybrid*. The main factor that distinguishes these geometries is the degree of flexibility they provide in the selection of neighbors and routes.

Neighbor selection refers to how the routing table entries of a peer are established, whereas route selection refers to how the next-hop can be determined in a routing process. Flexibility in the selection of neighbors and routes has a significant impact on the robustness and locality-awareness properties of the DHT-based system [32]. When allowing some freedom in the selection of neighbors and routes, one can choose neighbors and next routes, respectively, based on proximity. For instance, if the choice of neighbors is completely deterministic, it prevents the addition of features on top of the initial DHT proposal in order to achieve locality-aware routing tables. Further, flexible selections interfere in failures because they describe how many alternatives are there for the neighbor or the next-hop in case they are down. For instance, if there are no option for a next-hop, or only a few, this may destabilize or interrupt the routing process, which can greatly increase the number of hops or/and the latency.

In the following, we look at 4 geometries, tree, hypercube, ring and hybrid and discuss their flexibility degrees.

Tree. Peer IDs are the leaves of a binary tree of height $log\,N$, where N is the number of nodes in the tree (see Figure7). The responsible for a given key is the peer whose identifier has the highest number of prefix bits which are common with the key. The distance between any two peers is the height of their smallest common subtree. Each peer has $log\,N$ neighbors, such that the hth neighbor is at distance h from the peer. Let us consider the tree of height equal to 3 in Figure7. The peer with $ID = 010$ has the peer with $ID = 011$ as its 1st neighbor because their smallest common subtree is of height $h = 1$. Their IDs share a prefix of two bits and differ on the last bit. Similarly, the peer with $ID = 010$ has chosen the peer with $ID = 000$ as its 2nd because their smallest common subtree is of height $h = 2$. Their IDs share a prefix of one bit and differ on the two others. Routing is performed such that the prefix match between the target key and the ID of the intermediate peer is increased by one at each

[10]The routing state refers to the routing table of the peer.

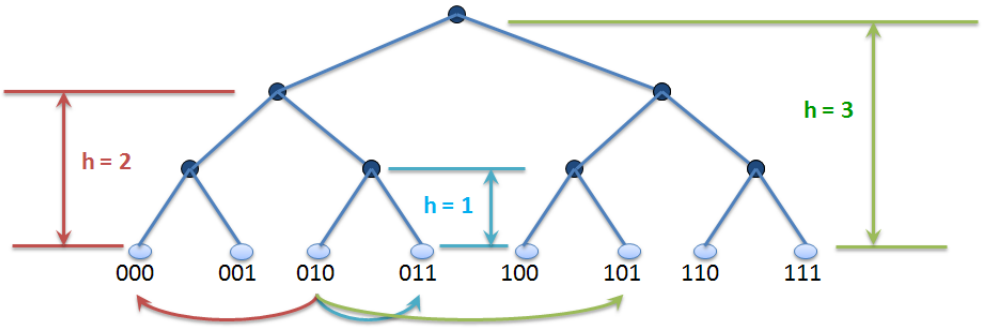


Figure 7: Tree routing geometry.

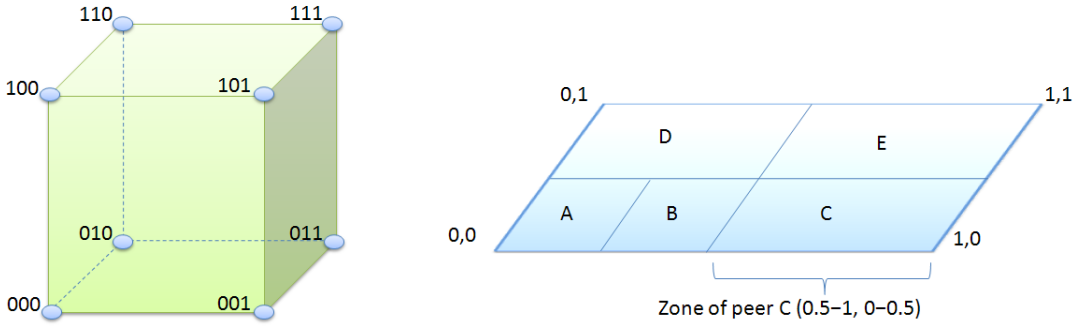
hop, until reaching the responsible peer. The well-known DHT implementation Tapestry [99] falls into this category.

The tree geometry gives a great deal of freedom to peers in choosing their neighbors; when choosing the ith neighbor, a peer has 2^{i-1} options. In the tree example, the peer with $ID = 010$ has $2^{2-1} = 2$ choices for its 2nd neighbor: the peer with $ID = 000$ and the peer with $ID = 001$ because they both belong to the subtree of height $h = 2$. However, this approach has no flexibility in the selection of routes: there is only one neighbor which the message must be forwarded to, i.e., this is the neighbor that has the most common prefix bits with the given key.

Hypercube. This geometry is based on a d-dimensional Cartesian coordinate space that is partitioned into a set of separate zones such that each peer is attributed one zone. Peers have unique identifiers with $log\, N$ bits, where N is the total number of peers of the hypercube. Each peer p has $log\, N$ neighbors such that the identifier of the ith neighbor and p differ only in the ith bit (see Figure8a). Query routing proceeds by greedily forwarding the given key via intermediate peers to the peer that has minimum bit difference with the key. Thus, it is somehow similar to routing on the tree. The difference is that the hypercube allows bit differences to be reduced in any order while with the tree, bit differences have to be reduced in strictly left-to-right order. CAN [69] uses a routing geometry similar to hypercubes. Figure8b shows a 2-dimensional $[0; 1] * [0; 1]$ coordinate space partitioned between 5 peers.

There are $(log\, N)!$ possible routes between two peers, which provides high route flexibility. However, each peer in the coordinate space does not have any choice over its neighbours coordinates since adjacent coordinate zones in the coordinate space cannot change. Therefore, the high route selection flexibility provided by Hypercubes is at the price of poor neighbor selection flexibility.

Ring. Peers are arranged in a one-dimensional cyclic identifier space and ordered clockwise with respect to their identifiers. Chord [89] represents the prototypical DHT ring. In Chord, each peer has an m-bit identifier, and the responsible for a key is the first peer whose identifier is equal to or greater that the



(a) Hypercube routing geometry.

(b) Hypercube-like structure of CAN.

Figure 8: Hypercube routing geometry.

key. Each peer p has $log\,N$ neighbors such that the ith neighbor has a distance from the peer clockwise in the circle equal to $2^{i-1} mod\, N$. Hence, any peer can route a given key to its responsible in $log\,N$ hops because each hop cuts the distance to the destination by half. In Figure9, the Chord peer with $ID = 8$ maintains 8 entries in its routing table called *finger table*.

Although Chord specifies the set of neighbors for each peer, the ring geometry does not necessarily needs such rigidity in neighbor selection. In fact, the $log\,N$ lookup bound is preserved as long as the ith neighbor is chosen from the range $[2^{i-1} mod\, N, 2^{i} mod\, N]$. This provides a great deal of neighbour selection flexibility because each peer would have 2^{i-1} options in selecting its ith neighbor. Moreover, to reach a destination, there are approximately $(log\,N)!$ possible routes. Therefore, the ring geometry also provides good route selection flexibility.

Hybrid. This geometry employs a combination of geometries. As a representative example, Pastry [76] is a popular DHT implementation that combines the tree and ring geometries, aiming at a locality-aware routing. To achieve this, each peer maintains a routing table, a leaf set, and a neighbourhood set. The routing table resembles the tree structure described previously, while the leaf set acts as the ring in routing. The neighbourhood set is used to maintain locality properties. During a lookup process, a peer uses first the tree structure represented by its routing table, and only falls-back to the ring via its leaf set if routing in the tree fails. This is why Pastry provides flexibility in neighbor selection, similar to the tree geometry. However, the matter is more sublte with respect to route selection flexibility. Given that a peer maintains an ordered leafset, it is able to take hops between peers with the same prefix (i.e., between branches of the tree) and still retain the bits that were fixed previously; this however does not necessarily preserve the $log\,N$ bound on the number of hops.

3.3.2 Behavior under Churn and Failures

Preserving the topology constraints is crucial to guarantee the correctness of lookup in structured overlays. However, churn and failures highly affect DHTs.

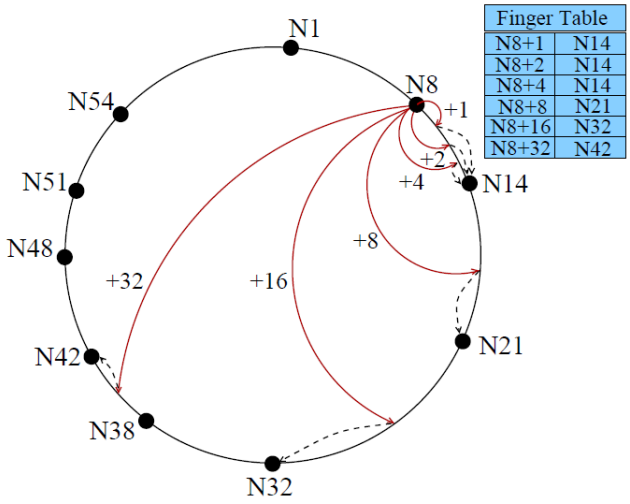


Figure 9: Ring routing geometry. Example of Chord with 10 peers and a peer ID of $m = 6$ bits.

When peer failures or leaves occur, they deplete the routing tables of the existing peers. Recovery algorithms are used to repopulate the routing tables with live peers, so that routing can continue unabated. The recovery can be reactive (upon detecting the failure of a peer referenced by the routing table) like in Pastry [76] or periodic (upon regular time intervals in the background) like in Chord [89]. Basically, the peer exchanges entries from the routing table with peers from its routing table and accordingly update its routing table. After a single peer leaves the network, most DHTs require $O(log\, N)$ repair operations, i.e., updates of routing tables affected by the leave (N is the total number of peers) . When a peer unexpectedly fails, the DHT needs more time and effort to first detect the failure and then repair the affected routing tables. It should also notify the application to take specific measures so that the content held by failed peers is not lost. Several approaches have been proposed to prevent this problem, most notably the replication of content at peers with IDs numerically close to the content's key [77].

When a new peer joins the overlay network, the DHT should detect the arrival and inform the application of the set of keys that the new peer is responsible for so that the relevant content is moved to its new home. Similarly to leaves and failures, the recovery algorithms should update the routing tables of the peers concerned by the new arrival.

However, if the churn rate is too high, the overhead caused by these repair operations can become dramatically high and could easily overwhelm peers [10].

Furthermore, recovery protocols take some time to repair and update the routing tables affected by joins and/or leaves. Given that new arrivals and departures are frequent in P2P environments, one must check the *static resilience* of a DHT [32], i.e., how well the DHT routing algorithm can perform before the overlay has recovered (before routing tables are restored and keys migrated

to new homes). DHTs with low static resilience require much faster recovery algorithms to be similarly robust. In such DHTs, requests that fail in routing should retry the lookup after a pause. A DHT with routing flexibility provides high static resilience because it has many alternate paths available to complete a lookup (see Section 3.3.1).

The analysis in [73] has examined the effects of churn on existing DHT implementations and derived two main observations. A DHT may either fail to complete a lookup, or return inconsistent answers (e.g., return the address of a peer that is no more responsible for the requested key). On the other hand, a DHT may continue to return consistent answers as churn rates increase, but it can suffer from a substantial increase in lookup latency.

3.3.3 Strengths and Weaknesses

Structured overlays offer strong guarantees on lookup efficiency while limiting the routing overhead. In particular, the ring topology is the best compromise that supports many of the properties we desire from such overlays. They have been used in a variety of applications such as content storage (e.g., OceanStore [49], Pastry [77]), multicast services (e.g., Bayeux [100], Scribe [78]) or large-scale query processing engines (e.g., Pier [35]).

However, two criticisms arise against these overlays and constitue major hurdle in the adoption of such systems. First, the tightly controlled topology requires high maintenance in order to cope with the frequent joins and leaves of peers. Moreover, studies [73] have shown that structured systems exhibit less than desirable performance under high churn rates because routing tables are affected and take time to be repaired. A second criticism concerns the limited flexibility provided by structured systems wrt. the autonomy of peers and the lookup functionality. Peers cannot freely choose their neighbors nor their responsabilities. Further, structured systems are designed in a way to provide key-based lookup which is convenient to exact-match queries. Their ability to support keyword searches and more complex queries is still an open issue.

Thus, structured overlays are the perfect match for applications that seek a scalable and guaranteed lookup but do not witness highly dynamic populations.

3.4 Requirements of P2P Systems

Based on this preliminary study on P2P systems, we observe that they introduce new requirements in respect of content sharing. The study in [18] identifies the following requirements:

- **Autonomy** defines the level of freedom granted to peers, mainly with respect to the placement of content. This is required to give peers proper incentives to cooperate. Indeed, it is usually not desired and rarely enabled to force storing content on peers.
- **Expressiveness** refers to the flexibility in query formulation. It should allow the user to describe the desired content at the level of detail that is appropriate to the target application.
- **Quality of service** has the most influence on user satisfaction. It can be defined with metrics like response time and hit ratio.

- **Efficiency** refers to the efficient use of resources of the P2P network (bandwidth, processing power, storage). Given the high rate of failures and churn, the maintenance protocol should neither compromise the gains with its overhead nor degrade the system performance. Also, efficiency implies that the routing protocol does not overload the network or the peers while not missing the available content.
- **Robustness** means that efficiency and quality of service are provided despite the occurrence of peer failures.
- **Security** is a major challenge given the open nature of P2P networks. With respect to content distribution, one of the most critical issues is the content authenticity which deals with the problem of distinguishing fake documents from original ones. We do not focus on this requirement in our study.

REQUIREMENTS	UNSTRUCTURED	STRUCTURED
Autonomy	free to choose neighbors and content	tight control on neighbors and content
Expressiveness	keywords	exact-match
Quality of service	no guarantees	deterministic
Efficiency	efficient maintenance	efficient lookup
Robustness	suitable for high churn	problems under high churn

Table 1: Comparison of P2P overlays.

Table 1 summarizes how the requirements are achieved by the two main classes of P2P networks. This is a rough comparison to understand the respective merits of each class. Obviously, there is room for improvement in each class of P2P networks. Regarding efficiency, structured systems provide a highly efficient lookup at the cost of a significant maintenance overhead, in opposition to unstructured systems.

Beyond this classical classification of P2P systems, there exist new trends in the P2P literature, that focus on other considerations and incur new challenges on the design of a P2P system. This is further investigated in Section 4.

4 Recent Trends for P2P Content Distribution

We have, so far, discussed P2P systems from a classical perspective. However, today's research is evolving towards more sophisticated issues about P2P systems, from the perspective of content distribution.

Recently, some have started to justify that unstructured and structured overlays are complementary, not competing. It is actually easy to demonstrate that depending on the application, one or the other type of overlay is clearly more adapted. In order to make use of the desirable features provided by each topology, there are efforts underway for combining both in the same P2P systems.

Further, the overlay can be refined through extracting and leveraging inherent structural patterns from P2P networks. These patterns can stem from the underlying physical network (e.g., physical proximity between peers) or be defined at the application layer (e.g., interest-based proximity between peers).

Matching the overlay with the underlying physical network greatly contributes in reducing communication and data transfer costs as well as user-perceived latencies. Additionally, leveraging interests of peers to organize them can ease the search for content and guide the routing of queries.

Another recent trend is the usage of gossip protocols as a mean to build and maintain the P2P overlay. Gossiping is also used to feed the overlay with indexing information in order to facilitate content search.

In the following, we present in more detail the aforementioned trends. In Section 4.0.1, we detail locality-based overlay matching and the existing solutions along these lines. Then, we present interest-based overlay matching in Section 4.0.2. In Section 4.0.3, we introduce the usage of gossip protocols in P2P systems. Finally, we review the existing approaches that combine several overlays in Section 4.0.4. Finally, we identify the major challenges to be met when aiming to achieve these new trends and accordingly discuss the aforementioned approaches.

4.0.1 Trend 1: Locality-Based Overlay Matching

As introduced in Section 3, the overlay topology defines application-level connections between peers and completely abstracts all features about the underlying physical network (e.g., IP level). In other terms, the neighborhood of a node is set without much knowledge of the underlying physical topology, causing a mismatch between the P2P overlay and the physical network. Figure4 clearly illustrates the mismatch between a P2P overlay and the underlying Internet. As an example, peer A has peer B as its overlay neighbor while peer C is its physical neighbor. This can lead to inefficient routing in the overlay because any application-level path from peer A towards the nearby peer C traverses distant peers.

More precisely, the scalability of a P2P system is ultimately determined by its efficient use of underlying resources. The topology mismatch problem imposes substancial load on the underlying network infrastructure, which can eventually limit the scalability [74]. Furthermore, it can severely deteriorate the performance of search and routing techniques, typically by incurring long latencies and excessive traffic. Indeed, many studies like [80] have revealed that the P2P traffic contributes the largest portion of the Internet traffic and acts as a leading consumer of Internet bandwidth. Thus, a fundamental challenge is to incorporate IP-level topological information in the construction of the overlay in order to improve routing performance. This topological information could also be used in the selection of close-by search results to ensure a good user experience. Topological information refers to locality-awareness because it aims at finding peers close in locality. Below, we present the main representative approaches that propose locality-based matching schemes.

Physical Clustering. In [48], clustering has been used to group physically close peers into clusters. The approach relies on a centralized engine to identify clusters of close peers under common administrative control. To achieve this, the central server uses IP-level routing information which is not directly available to end-user applications. Thus, the main drawbacks of this approach are the centralized topology control and the topological information itself, which prevents it from being scalable and robust to churn.

In the context of application-level multicast and media streaming, many solutions aim at constructing a locality-aware overlay because of the strong requirements on the delivery quality. The NICE protocol [2] builds a hierarchy of clusters rooted at the source, whith close peers belonging to the same part of the hierarchy. However, maintaining the hierarchy under churn may incur high overhead and affect performance.

LTM Technique. The LTM (*Location-aware Topology Matching*) technique [53] targets unstructured overlays. It dynamically adapts connections between peers in a completely decentralized way. Each peer issues a detector in a small region so that the peers receiving the detector can record the relative delay. Accordingly, a receiving peer can detect and cut most of the inefficient logical links and add closer peers as neighbors. However, this scheme operates on long-time scales where the overlay is slowly improved over time.Given that participants join and leave on short time-scales, a solution that operates on long-time scales would be continulally reacting to fluctuating peer membership without stabilizing.

Locality-Aware Structured Overlays. While the original versions of structured overlays did not take locality-awareness into account, almost all of the recent versions make some attempt to deal with this primary issue. [71] identifies three main approaches.

- *Geographic layout*: the peer IDs are assigned in a manner that ensures that peers that are close in the physical network are close in the peer ID space.
- *Proximity routing*: the routing tables are built without locality-awareness but the routing algorithm aims at selecting, at each hop, the nearest peer among the ones in the routing table. For this, flexibility in routing selection is required as layed out in Section 3.3.1.
- *Proximity neighbor selection*: the construction of routing tables takes locality-awareness into account. When several candidate peers are available for a routing table entry, a peer prefers the one that is close in locality. To achieve this, flexibility in neighbor selection is required as pointed out in Section 3.3.1.

Pastry [76] and Tapestry [99] adopt proximity neighbor selection. In order to preferentially select peers and fill routing tables, these systems assume the existence of a function (e.g., round-trip-time RTT) that allows each peer to determine the physical distance between itself and any another peer. Although this solution leads to much shorter query routes, it requires expensive maintenance mechanisms under churn. As peers arrive and leave, routing tables should be repaired and updated. Without timely repairing, the overlay topology will diverge from optimal condition as inefficient routes gradually accumulate in routing tables.

A design improvement [70] of CAN aims at achieving geographic layout. It relies on a set of well-known landmarks spread across the network. A peer measures its round-trip time (RTT) to the set of landmarks and orders them by increasing latency (i.e., network distance). The logical space of CAN is then

divided into bins such that each possible landmarks ordering is represented by a bin. Physically close nodes are likely to have the same ordering and hence will belong to the same bin. This is illustrated in Figure 10. We have 3 landmarks (i.e., L1, L2, and L3) and, accordingly, the CAN coordinate space is divided into 6 bins (3! = 6). Since peers N1, N2, and N3 are physically close (see Figure 10 (a)), such peers produce the same landmark ordering, i.e., L3¡L1¡ L2. As a result, N1, N2, and N3 are placed in the same bin of the overlay network, and they take distinct neighbor zones (see Figure 10 (b)). The same approach applies to other peers. Notice that such approach is not perfect. For instance, peer N10 is closer to N3 than N5 in the physical network whereas the opposite situation is observed in the overlay network. Despite its limited accuracy, this technique achieves fast results and copes well with dynamicity. In addition, binning has the advantage of being simple to implement and scalable since peers independently discover their bins without communicating with other participants. Furthermore, it does not incur high load on the landmark machines: they need only echo ping messages and do not actively initiate measurements nor manage measurement information. To achieve more scalability, multiple close-by nodes can act as a single logical landmark.

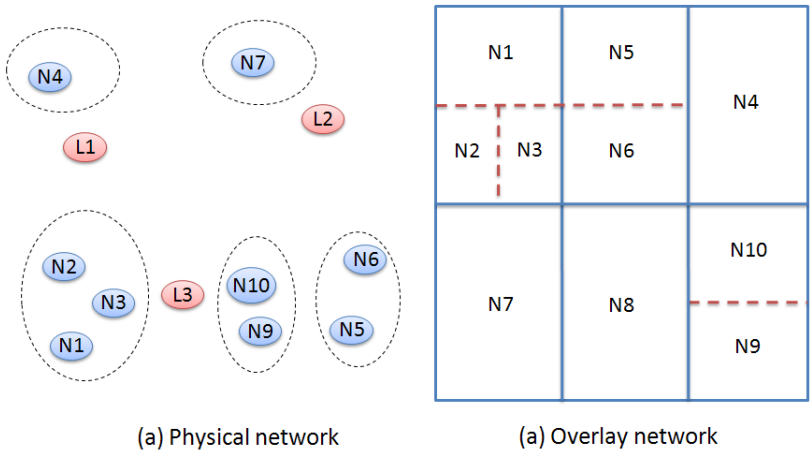


Figure 10: Locality-aware construction of CAN

An observation about the aforementioned locality-aware schemes is that the technique used in Pastry and Tapestry is very protocol-dependent and thereby cannot be extended to other contexts in a straigthforward manner, whereas the binning technique can be more generally applied in contexts other than in structured overlay, like unstructured overlays.

4.0.2 Trend 2: Interest-Based Topology Matching

In attempt to improve the performance of P2P systems and the efficiency of search mechanisms, some works have addressed the arbitrary neighborhood of peers from a semantic perspective. Recent measurement studies [33, 28, 85] of

P2P workloads have demonstrated the inherent presence of *semantic proximity* between peers, i.e., similar interests between peers. They have shown that exploiting the implicit interest-based relationships between peers may lead to improvements in the search process. In short, they have reached the following conclusion: "if a peer has an object that I am interested in, it is very likely that he will have other objects that I am (or will be) interested in".

These interest-based relationships can be translated into logical connections between peers that can either replace or be added on top of a peer neighborhood. If we consider Figure 4, peer A has peer B as a neighbor specified by its overlay topology and could have extra connections with semantically similar peers like peer D. Then, these semantic connections can be used to achieve efficient search.

In the following, we discuss two representative works along these lines. They were initially proposed for unstructured overlays. When applying one of them, a peer maintains two types of neighbors: its neighbors in the unsructured overlay (e.g., random peers) and its interest-based neighbors. Upon receiving a query, the peer uses its interest-based neighbors first; if this first phase fails the normal search phase is performed via its normal neighbors. In superpeer overlays, the first phase can be used to bypass the superpeers thus alleviating their load.

Semantic Clustering. Garcia-Molina et al. [16] introduces the concept of semantic overlays and advocates their potential performance improvement. Peers with semantically similar content are grouped into clusters together. Clusters can overlap because a peer can simultaneously belong to several clusters related to its content. To achieve this, the authors assume global knowledge of the semantic grouping of the shared documents and accordingly choose a predefined classification hierarchy. Then, each peer decides which clusters to join by classifying its documents against this hierarchy. To join its clusters, the peer finds peers belonging to these clusters by flooding the network. However, It is not clear how this solution performs in the presence of dynamic user preferences.

Interest-Based Shortcuts. In [85], the concept of shortcut is proposed, allowing peers to add direct connections to peers of similar interests besides their neighbors. The similarity of interests are captured implicitly based on recent downloads and accordingly, interest-based shortcuts are dynamically created in the network: basically, a peer adds shortcuts to peers among those from which it had recently downloaded content. In practice, these shortcuts are discovered progressively while searching for content via flooding. Furthermore, the time for building interest-based groups is non-trivial, and these groups may be no more useful when the peer goes offline and then online again, due to the dynamic nature of P2P networks.

The aforementioned schemes may also be applied to structured overlays. In addition to its routing table, a peer may maintain interest-based neighbors and use them conjunctly. However, this increases the routing state at each peer and incurs extra storage and update overhead.

4.0.3 Trend 3: Gossip Protocols as Tools

We now present the usage of gossip protocols in P2P systems. They can serve as efficient tools to achieve new P2P trends in a scalable and robust manner.

Gossip has recently received considerable attention from researchers in the field of P2P systems [47]. In addition to their inherent scalability, they are simple to implement, robust and resilient to failures. They are designed to deal with continuous changes in the system, while they exhibit reliability despite peer failures and message loss. This makes them ideally suited for large-scale and dynamic environments like P2P systems. In this section, we provide generic definition and description of gossip protocols, then we investigate how P2P systems can leverage these protocols.

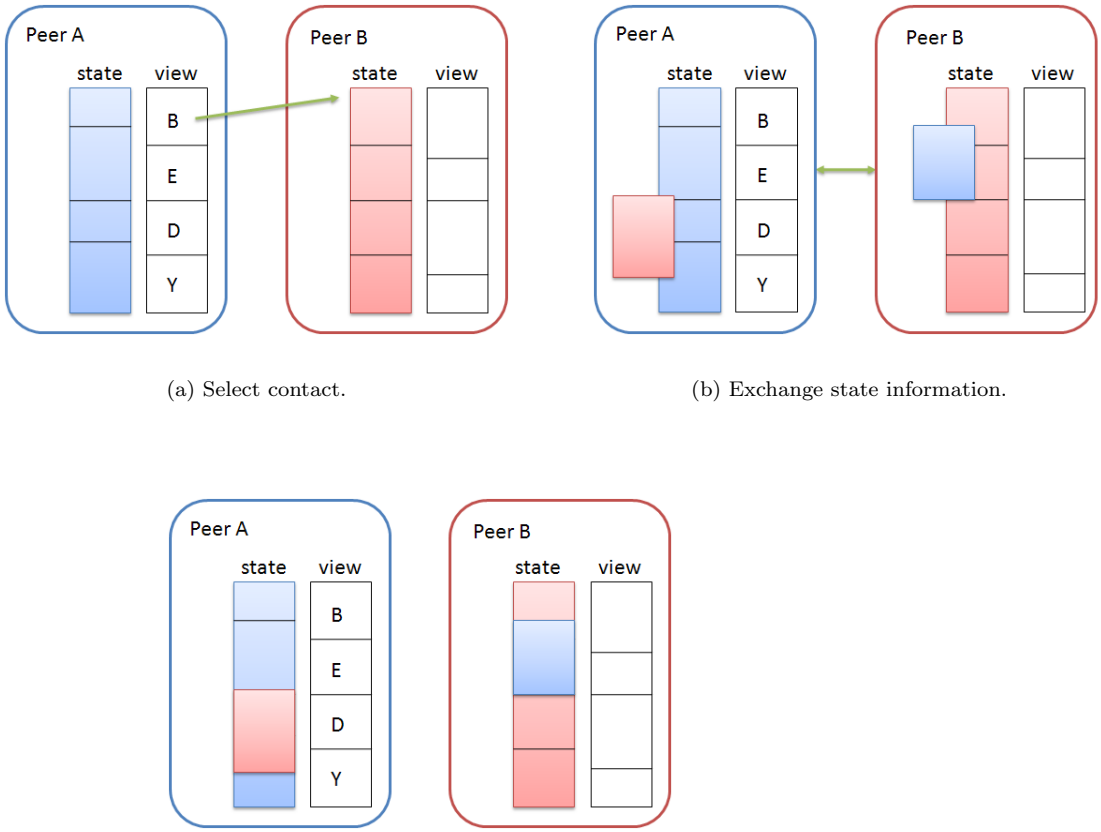
Generic Definition Gossip algorithms mimic rumor mongering in real life. Just as people pass on a rumor by gossiping to their contacts, each peer in a distributed system relays new information it has received to selected peers which in their turn, forward the information to other peers, and so on. They are also known as *epidemic protocols* in reference to virus spreading [19].

Generic Algorithm Description The generic gossip behavior of each peer can be modeled by means of two separate threads: an *active thread* which takes the initiative to communication, and a *passive thread* which reacts to incoming initiatives [47]. Peers communicate to exchange information that depends strictly on the application. The information exchange can be performed via two strategies : *push* and *pull*. A push occurs in the active thread, i.e., the peer that initiates gossiping shares its information upon contacting the remote peer. A pull occurs in the passive thread, i.e., the peer shares its information upon being contacted by the initiating peer. A gossip protocol can either adopt one of these strategies or the combination of both (i.e., *push-pull* which implies a mutual exchange of information during each gossip communication).

Figure 11 illustrates in more detail a generic gossip exchange. Each peer A knows a group of other peers or *contacts* and stores pointers to them in its *view*. Also, A locally maintains information denoted as its *state*. Periodically, A selects a contact B from its view to initiate a gossip communication. In a pull-push scheme, A selects some of its information and sends them to B which, in its turn, does the same. Upon receiving the remote information, each one of A and B merges it with its local information and update their state. At that point, how a peer deals with the received information and accordingly update its local state is highly application dependent.

How P2P Systems Leverage Gossip Protocols Gossip stands as a tool to achieve 4 main purposes [47]: *dissemination*, *resource monitoring*, *topology construction* and *peer sampling*. Figure 12 illustrates these gossip-based services and how they interfere in a P2P system that is represented by an overlay layer and a search layer.

Introduced by Demers et al. [19], **dissemination** has traditionally been the purpose of gossiping. In short, the aim [26] is to spread some new information throughout the newtork by letting peers forward messages to each other. The information gets propagated exponentially through the network. In general, it takes $O(logN)$ rounds to reach all peers, where N is the number of peers. Figure 12 shows that gossip-based dissemination can be used to feed the search layer with indexing information useful to route queries. Basically, a peer can



(a) Select contact.

(b) Exchange state information.

(c) Merge and update local state.

Figure 11: Peer A gossiping to Peer B.

maintain and gossip information about the content stored by other peers and decide accordingly to which peers it should send a query.

Then, gossiping has turned out to be a vehicule of **resource monitoring** in highly dynamic environments. It can be used to detect peer failures [72], where each peer is in charge of monitoring its contacts, thus ensuring a fair balance of the monitoring cost. Further, gossip-based monitoring can guarantee that no node is left unattended, resulting in a robust self-monitoring system. In Figure 12, the monitoring service is used to maintain the overlay under churn by monitoring a peer's neighbors. In addition, it interferes in the search layer to monitor indexing information in face of content updates and peer failures.

Recently, various researches have explored gossiping as a mean for **overlay construction and maintenance** according to certain desirable topologies (e.g., interest-based, locality-based, random graphs), without requiring any global information or centralized administration. In such systems, peers self-organize under the target topology, via a selection function that determines which neighbors are optimal for each peer (e.g., semantic or physical proxim-

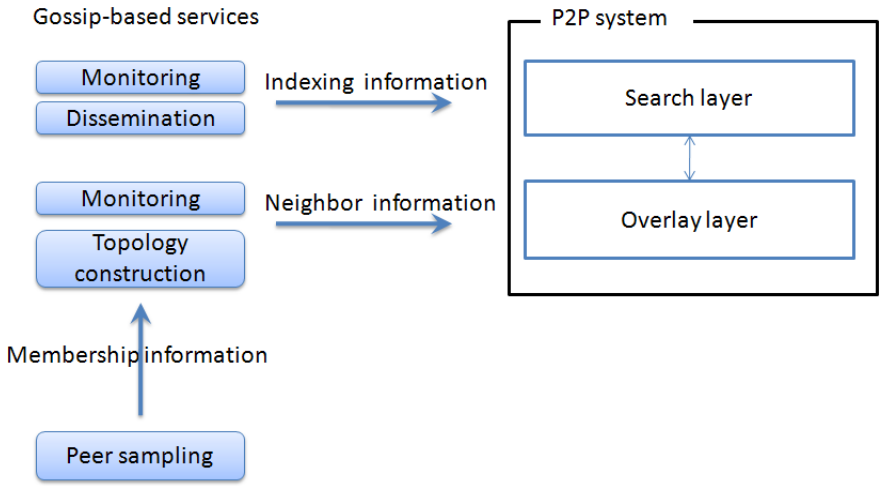


Figure 12: How a P2P system can leverage gossiping.

ity). Along these lines, several protocols have been proposed such as Vicinity[94] which creates a semantic overlay and T-Man[39] that provides a general framework for creating topologies according to some ranking function. Figure 12 represents the topology construction service providing peers with specific neighbors and thereby connecting the P2P overlay.

Analyses [40] of gossip protocols reveal a high reliability and efficiency, under the assumption that the peers to send gossip messages to are selected uniformly at random from the set of all participant peers. This requires that a peer knows every other peer, i.e., that the peer has *global knowledge of the membership*, which is not feasible in a dynamic and large-scale P2P environment. **Peer sampling** offers a scalable and efficient alternative that continuously supplies a node with new and random samples of peers. This is achieved by gossiping membership information itself which is represented by the set of contacts in a peer's view. Basically, peers exchange their view information, thus discovering new contacts and accordingly updating their views. In order to preferentially select peers as neighbors, gossip-based overlay construction may be layered on top of a peer sampling service that returns uniformly and randomly selected peers. Well-known protocols of peer sampling are Lpbcast, Newscast and Cyclon[93]. In Figure 12, we can see the peer sampling service supporting other gossip-based services and supplying them with samples of peers from the network.

To conclude this section on gossip protocols, we shed light on their salient strenghts and weaknesses.

Strengths Gossip algorithms have the advantage of being extremely simple to implement and configure [3]. Furthermore, they perfectly meet the decentralization requirement of P2P systems since many of them are designed in a way

to let peers take local-only decisions. If properly designed, they can balance and limit the loads over participant peers.

Gossiping also provides high robustness which stems from the repeated probabilistic exchange of information between two peers [47]. Probabilistic choice refers to the choice of peer pairs that communicate while repetition refers to the endless process of choosing two peers to exchange information. Therefore, gossip protocols are resilient to failures and frequent changes and they cope well with the dynamic changes in P2P systems.

Weaknesses The usage of gossip might introduce serious limitations [3]. The protocol running times can be slow and potentially costly in terms of messages exchanged. One should carefully tune gossip parameters (e.g., periodicity) in a way that matches the goals of the target appplication.

4.0.4 Trend 4: P2P Overlay Combination

We now present a recent trend that is changing the classical categorization of P2P systems. Lately, several approaches have been proposed to build a P2P system over multiple overlays in order to combine their functionalities and leverage their advantages. The combination might involve structured and unstrucured overlays as well as interest- (or semantic) and locality-based overlays. The construction and maintenance of the combined overlays might imply additional overhead which should not compromise the desirable gains. Below, we present and discuss some examplary approaches.

Structured & Unstructured. The approach presented in [9] improves the unstructured Gnutella network by adding some structural components. The motivation behind is that unstructured routing mechanisms can support complex queries but generate significant message overhead. Structella [9] replaces the random graph of Gnutella with the structured overlay of Pastry, while retaining the flexible content placement of unstructured P2P systems. Queries in Structella are propagated using either flooding or random walks. A peer maintains and uses its structured routing table to flood a query to its neighbors, thus ensuring that peers are visited only once during a query and avoiding duplicate messages. However, this work does not enable the important features of locality and interest awareness.

Interest & Locality-based. The work in [7] builds Foreseer, a P2P system that combines an interest-aware overlay and a locality-aware overlay. Thus, each peer has two bounded sets of neighbors: proximity-based (called *neighbors*) and interest-based (called *friends*). Finding neighbors relies on a very basic algorithm that improves locality-awareness slowly with time. Whenever a node discovers new peers, it replaces its neighbors with the ones that are closer in latency. A similar scheme is used to progressively make and refine friends from the peers that satisfy queries of the node in question. Friends are preferentially selected by comparing their content similarity with the target node. However these schemes operate on long-time scale.

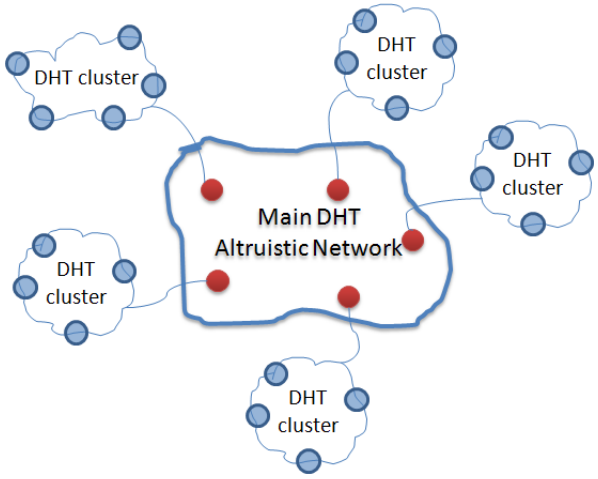


Figure 13: A two-layers DHT overlay [60].

Joint Overlay. In [55], the authors leverage the idea of cohabiting several P2P overlays on a same network, so that the best overlay could be chosen depending on the application. The distinctive feature of this proposal is that, in the joint overlay, the cohabiting overlays share information to reduce their maintenance cost while keeping the same level of performance. As an example, they describe the creation of a joint overlay with a structured overlay and an interest-based unstructured overlay using gossip protocols. Thus each peer belongs to both overlays and can alternatively use them.

DHT Layering or Hierarchy. The work in [60] organises the structured overlay in multiple layers in order to improve performance under high levels of churn. They introduce the concept of heterogeneity with respect to peer behavior, being altruistic or selfish. The idea is to concentrate most routing chores at altruistic peers; these peers are willing to carry extra load and have the required capabilities to do so. The authors also assume that altruistic peers stay connected more than others. Thus, a main structured overlay is built over altruistic peers, and each one in its turn is connected to a smaller structured overlay of less altruistic peers. Figure 13 shows an example of a two-layers DHT, where the main DHT represents the altruistic network and links several DHT-structured clusters. The P2P overlay can be further clustered, resulting into multiple layers.

A similar work is proposed in [81] and addesses the problem of load balancing in a heterogenous environment in terms of capacities. Likewise, a main structured overlay is built over high-capacity peers, and each one acts as a super-peer for a locality-based cluster of regular peers. Each peer has an ID obtained by hashing its locality information (using the binning technique of Section 4.0.1). A regular peer is assigned to a super-peer whose ID is closest to the peer's ID, which results in regular peers being connected to their physically closest super-peer.

These proposals are orthogonal to our work, as they mainly focus on DHTs performance under heterogeneity. It is not clear how they can support content distribution and location.

4.0.5 Challenges to Be Met

When refining the P2P network via sophisticated techniques (like locality or interest aware schemes), one should make sure that the overhead is worth the performance improvement. Based on the aforementioned trends, we have identified two major **challenges** that need to be explored:

Challenge 1 *To capture or gather the information (e.g., topological or semantic relationships) in a manner that is both practical and scalable. This should be done without:*

- *requiring global knowledge or centralized administration.*
- *incurring large overheads of messages and/or data transfers on the existing overlay.*

Challenge 2 *To be adaptive to dynamic changes and churn. Indeed, the solution should provide a scheme that can still be valid and effective when new peers join or/and existing ones leave. For this, it should:*

- *operate on short-time scales. Given that participants join and leave on short time-scales, a solution that operates on long-time scales would be continulally reacting to fluctuating peer membership without stabilizing.*
- *avoid grouping peers into a static configuration which does not evolve well as the behavior or characteristics of peers change.*

4.0.6 Discussion

In this section, we have reviewed the recent trends in the P2P literature, mainly from the perspective of content sharing. We have seen that they improve the performance of P2P infrastructures but incur additional challenges related to scalability and dynamicity on their design. Table 2 summarizes the main approaches that integrate recent P2P trends and evaluates them with respect to the challenges.

In short, matching the overlay with a locality or interest-aware scheme could bring great benefits to the P2P system in terms of efficiency and quality of service. However, the schemes should be kept simple and practical. Among the proposed approach, the binning technique is the perfect match to achieve locality-awareness with respect to the challenges. It relies on topological information that is practical and incurs limited overhead (Challenge 1); it also operates fast and can easily adapt to changes (Challenge 2).

Gossiping can be used to build locality and interest-based schemes and can answer the challenges. It can be designed in a way that provides simplicity, decentralization and high robustness. In general, gossip is a tool, not an end in itself. It should be used selectively, in contexts where gossip is the best choice, mainly in the fields of monitoring and dissemination and overlay maintenance. This implies that gossip protocols need to be combined with other tools to build

Trend	Challenge 1	Challenge 2
Locality-aware schemes		
Physical clustering	no	no
LTM technique	yes	no
Pastry & Tapestry locality-aware scheme	yes	no
Binning technique	yes	yes
Foreseer	yes	no
Interest-aware schemes		
Semantic clustering	no	no
Interest-based shortcuts	yes	no
Foreseer	yes	no
Using Gossip	partially	yes
Joint Overlay	no	no

Table 2: Trends vs. challenges.

an efficient P2P infrastructure [3]. Further, efficient tuning is needed so that gossip does not incur significant delays and overheads in terms of messages. The message overhead might prevent gossip protocols from fully satisfying Challenge 1.

Finally, we have concluded that structured and unstructured overlays should not be seen as competing but rather complementing each other. Each category provides specific and unique functionalities. Combining different overlays and schemes might reveal interesting, yet very challenging. In particular, the maintenance of several overlays should not overwhelm the P2P system. An interesting solution is to leverage the combination in the maintenance mechanisms (e.g., exploiting one overlay to maintain the other). Also, gossip can be a potentially effective solution for this issue that requires no centralization if properly designed.

5 P2P Content Distribution Systems

In the previous section, we provided a generic presentation of P2P systems which can serve as infrastructures for applications like content distribution. In this section, we deepen our study on P2P content distribution systems. In particular, we examine the existing proposals and identify the shortcomings according to the requirements and challenges identified through this report.

Most of the current P2P applications fall within the category of content distribution, which range from simple file sharing, to more sophisticated systems that create a distributed infrastructure for organizing, indexing, searching and retrieving content [1]. P2P content distribution functionalities are achieved via collaboration among peers, scalability being ensured by resource sharing. By distributing tasks across all participating peers, they can collectively carry out large-scale content distribution without the need for powerful and dedicated servers.

In the following, we first give an overview (Section 5.1) where we define the context of P2P content distribution and recall the P2P and CDN requirements

discussed in the previous sections. Then, we discuss the existing works and enlighten the open issues of P2P file sharing (Section 5.2) and P2P CDN (Section 5.3).

5.1 Overview

Recall that the design of a CDN brings stringent requirements which are *performance*, *reliability* and *scalability* (cf. Section 2.3). In contrast to traditional CDNs, a P2P infrastructure relies on peers which are not dedicated servers but autonomous and volunteer participants with their own heterogenous interests. When building a CDN over a P2P infrastructure, it is vital to reconcile and coordinate these requirements with the ones introduced by P2P systems, i.e., *autonomy, expressiveness, efficiency, quality of service, robustness,* and *security* (cf. Section 3.4). Let us recapitulate and identify the different correlations between CDN and P2P requirements. Further, we point out where the P2P recent challenges interfere.

Performance, Quality of Service. Performance meets the requirement of quality of service. It is ensured via locality-aware and efficient location of content as laid out previously. While many P2P systems abstract any topological information about the underlying network, locality-awareness should be a top priority in order to achieve short query response times. The locality-aware solution should overcome Challenges 1 and 2, i.e., it should be kept simple, incur acceptable overhead, operate fast and adapt to churn and high scales.

Scalability, Efficiency. In order to make efficient use of P2P inherent scalability, it is essential to distribute load equitably over peers. This is realized if all peers fairly share the processing of queries as well as the routing load. However, when some peers hold popular content, they may present hot spots, attracting large amounts of queries.

Reliability/Robustness, Autonomy. Reliability can only be ensured by the robustness of the P2P system under the dynamic nature of its peers. There is a strong correlation between robustness and autonomy [18]. Indeed, churn and failure rates are much higher that in CDN infrastructures because of the autonomous nature of peers. Routing and serving querie can be difficult to achieve as peers join and leave frequently and unexpectedly. Furthermore, the solutions of caching and replication that improve content availability are highly constrained by the *autonomy* of peers.

Efficiency, Autonomy. Decoupling efficiency from autonomy seems to be very challenging, given that most existing techniques tend to sacrifice autonomy to achieve efficiency [18]. This is because less autonomy allows more control on the content placement and topology such that there exist a deterministic way to locate content within bounded cost. In addition, search seems to be more efficient if the content is replicated. An interest-based scheme might be useful to leverage the interests of peers in the search and replication mechanisms. To be efficient, the scheme must meet Challenges 1 and 2 by being dynamic, practical

and scalable.

Before we depen our analysis of P2P content distribution, let us stand back and get an overview of the context. Figure 14 illustrates the relation between the P2P infrastructure and content distribution as its overlying application. The P2P infrastructure provides specific services which are identified by [1] as follows: routing and location, anonymity and reputation management. We focus on P2P infrastructures for routing and location. The operation of any P2P content distribution system relies on a network of peers within which messages must be routed with fault-tolerance and minimum overhead, and through which peers and content can be efficiently located. We have previously seen different infrastructures and algorithms that have been developed to provide such services. As laid out in the previous sections, the infrastructure characteristics, i.e., the topology, the routing protocol, the degree of centralization and structure, play a crucial role in the performance, reliability and scalability of the P2P content distribution system. Figure 14 shows that the application layer contains functionalities that are specifically tailored to build content distribution. Among these functionalities, we mention indexing, replication and caching which will be discussed along the next sections.

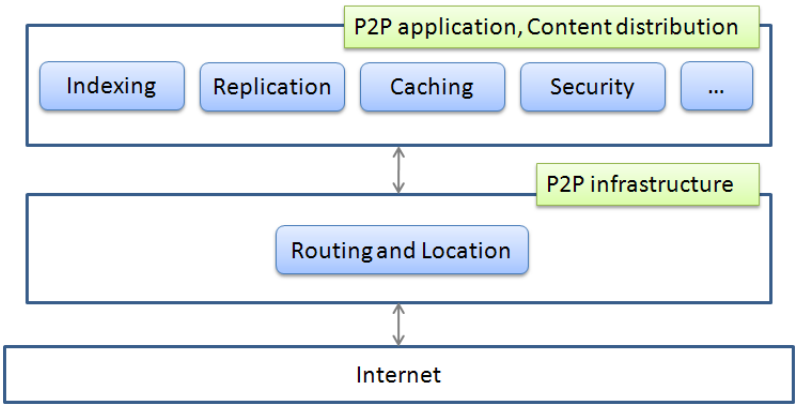


Figure 14: P2P infrastructure for content distribution.

5.2 P2P File Sharing

File sharing remains widespread in P2P networks. Some of the most popular networks are BitTorrent, FastTrack/Kazaa, Gnutella, and eDonkey. They are generally deployed over unstructured overlays, mainly due to their flexibility and support for keyword search.

File-sharing applications can afford to have looser garantees on the CDN requirements because such applications are meant for a wide range of users from non-cooperating environments [98]. These are typically light-weight applications that adopt a best-effort approach to distribute content and yet are accepted by

the user population [1]. Nonetheless, these systems should rigorously aim at keeping the network load at bay to enable a deployment over large-scales.

Since unstructured networks commonly use blind techniques to locate files as discussed in Section 3.2.2, many efforts have been made to avoid the large volume of unnecessary traffic incurred by such techniques. As such, *informed techniques* have been proposed, which rely on additional information about object locations to route queries. Typically, a peer can maintain an index of the content provided by other peers and decide accordingly to which peers it should send the query.

Next, we provide more insight into P2P file sharing systems by identifying their inherent properties. Then, we discuss the indexing techniques proposed in this context.

5.2.1 Inherent Properties

P2P file sharing exhibit inherent properties that should be well understood in order to design efficient solutions. In a nutshell, they are characterized by a high level of *temporal locality* in queries, involve a *natural replication* of files, and commonly witness keyword queries. Furthermore, P2P file sharing systems are considered as the leading consumer of Internet bandwidth [80]. The challenges are thereby to leverage intrinsic aspects (natural replication, temporal locality) and address inherent issues (keyword lookup, bandwidth consumption).

Natural Replication. File sharing systems vehiculate a "natural" replication of files, which is enabled by the flexibility of unstructured overlays with respect to content placement. When a peer requests a file, it downloads a copy which is often made available for upload to other peers. Thus, the more popular a file, the more it is "naturally" replicated and spread into the P2P network [10, 31].

Temporal Locality. Several analyses [56, 50, 84] of P2P file sharing systems observed that the queries exhibit significant amounts of *temporal locality*, that is, queries tend to be frequently and repeatedly submitted, requesting few popular files. Accordingly, they advocated the potential of caching to capitalize on this temporal locality. Caching is often done for the purposes of improved performance (i.e., higher hit ratio, reduced latencies and bandwidth costs). It can also be viewed as a cost-effective version of replication since it takes advantage of the unused storage resources and can evict copies at any time.

Keyword Lookup. File-sharing systems like Gnutella [30] vehiculate a simple keyword match. Users often generate queries that contain a set of keywords and peers generate query reponses referring to files whose names contain all the query keywords. Thus, query routing are required to support keyword lookup.

Bandwidth Consumption. Many measurement studies on P2P file sharing (e.g., [44]) shed light on the tremendous bandwidth consumption and its detrimental impact on both users and Internet Service Providers (ISPs). For the end users, their participation into a P2P network swamps all the available bandwidth and renders the link ineffective for any other use. For the ISPs, the P2P traffic is a major source of costs since an ISP handles the file tranfer at

the physical network layer. This increase of costs on ISPs is passed on to the user in the form of higher prices to access the Internet. The main reason behind this pertinent problem involves file transfers. P2P files are three orders of magnitude larger than web objects, since the majority of shared files are audio and movies [50, 80]. Also, they are randomly transferred between peers without any consideration of network distances.

On the one hand, the studies suggest to cache files in order to remedy this problem. However, this cannot be achieved without relying on a dedicated caching infrastructure. Indeed, in file sharing communities, users rarely accept to store or cache large files on behalf of each others.

On the other hand, the studies of [31, 44] present evidence on the potential bandwidth savings of locality-aware file transfers. Indeed, the analysis in [31] has shown that there is an untapped locality in file-sharing workload, i.e., a requested file is likely to be available at peers close to the requester in network locality. This means that there is substancial opportunity to improve file sharing performance by exploiting the untapped locality. In short, a query can be intentionally redirected towards nearby files, to optimize the file transfer.

BitTorrent [66] addresses this issue uner a different angle. Basically, a peer downloads multiple fragments in parallel from multiple downloaders of the target file, thus distributing the load among several peers. There are two primary concerns about this approach. First, it ignores locality-awareness by randomly choosing downloaders, which can further accentuate the bandwidth problem. The second concern is the centralized aspects of the search operation which limits its scalability and robustnees. To share a file, a peer first creates a metadata file called a *torrent* that contains information about the *tracker*. A tracker is a centralized server that keeps track of all current downloaders of a file and coordinates the file distribution. Peers that want to download the file must first obtain a torrent file for it, and connect to the specified tracker, which tells them from which other peers to download the fragments of the file. BitTorrent provides no way to index torrent files which are thus hosted by specific websites. On-going improvements aim at distributing the tracker's functionality (i.e., the discovery of file downloaders) over the peers via DHT or gossip protocols. BitTorrent can also serve as a P2P CDN to distribute web content and relieve original web servers.

5.2.2 Indexing Approaches

In unstructured networks, informed search is achieved by the use of distributed indexes to route queries. Basically, a peer maintains indexes related to the content stored by remote peers. Then, the peer evaluates any received query against its indexes and redirects it to peers that can contribute to it.

In general, an index yields a trade-off between compactness and accuracy, and between maintenance overhead and broadness. The more the index compactly represents the content, more storage efficiency is achieved but more false positives can result from index lookup. At the same time, the more broad is the coverage of the index (i.e, indexing distant content), the more effort and overhead is generated to maintain the indexes in dynamic environments.

An indexing approach should overcome several challenges, so that it does not partially offset the benefits of indexing itself. Below, we identify three main challenges:

- limit the overhead involved in the creation and update of indexes.
- support keyword search.
- introduce locality-awareness.

There are two types of indexes, a *forwarding index* and a *location index*. A *forwarding index* allows to reach the requested object within a varying number of hops, while a *location index* allows to reach the target in a single hop.

The approaches of **forwarding indexes** suply direction information towards the content, rather than its actual location. Two representative approaches are *routing indices* and *local indices*.

Routing Indices. This technique [15] assumes that all documents fall into a number of topics, and that queries request documents on particular topics. Also, each peer stores, for every topic, the approximate number of documents that can be retrieved through each one of its neighbors (i.e., including all the peers accessible from or linked to this neighbor). Figure15 illustrates the use of routing indices (RI) over four topics of interest. Considering the RI maintained by peer A, the first row contains the summary of its local index, showing that A has 300 documents (30 about databases, 80 about networks, none about theory, and 10 about languages). The rest of the rows represent compound RI. For example, they show that peer A can access 100 database documents through D (60 in D, 25 in I, and 15 in J).

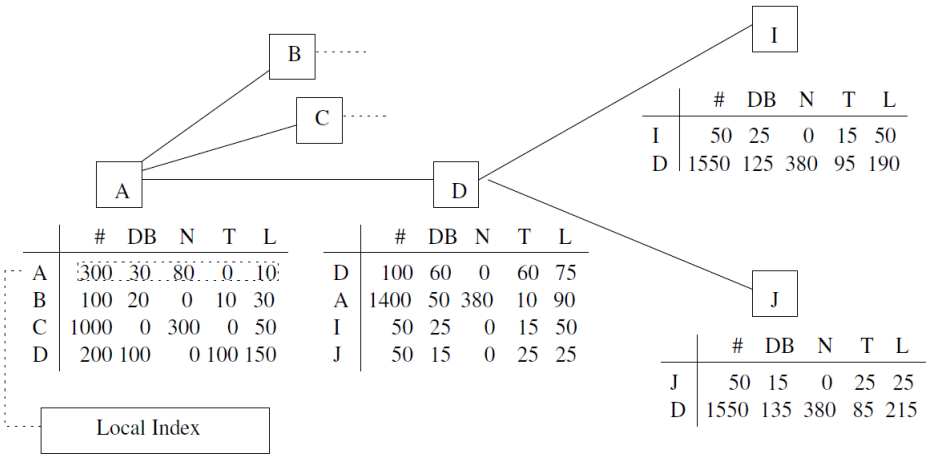


Figure 15: Example of routing indices [15].

Local Indices. This approach is proposed in [98]. Each peer maintains an index over the content of all peers within r hops of itself, and can therefore process any received query on behalf of these peers. While a query is routed using BFS (breadth-first-search or flooding), it is processed only at the peers

that are at predefined hop distances from the query originator. According to the authors' analysis, the hop distance between two consecutive peers that process the query must be $2*r+1$. This allows querying all content without any overlap and reducing the query processing time.

In the aforementioned appoaches, the indexes maintained by each peer would be extraordinarily large, and hence the overhead related to their creation and update may become prohibitively expensive, thus compromising the benefits. Furthermore, they do not consider locality-awareness for the purpose of reducing bandwidth consumption.

Another category of approaches uses **location indexes** which aim at determining which peers can provide certain content. Examples of such approaches are *intelligent BFS*, *Bloom Filter-based indices* and *index caching*.

Intelligent BFS. This technique [43] adapts the basic BFS algorithm. A peer maintains for each neighbor the list of recently answered queries from (or through) this neighbor. When a peer receives a query, it identifies all listed queries that are similar to the newly received query based on some similarity metric, and sends the query to the neighbors that have returned most answers for the similar queries. If an answer is found for the query at a peer, a message is sent to the peers over the reverse path in order to update their statistics. However, this technique produces more routing messages because of update messages. In addition, it can not be easily adapted to the peer departures and file deletions, and it ignores locality-aware aspects.

Bloom Filter-Based Indices. Bloom filters [4] have long been used as a lossy summary technique. The works in [11, 7] use a Bloom filter to represent the collection of keywords that characterize the objects shared by a peer. By first examining the filter, one can see if a queried file might be at the peer before actually searching the local repository of the peer. Thus, a peer selectively forwards a query to the peers that might satisfy the query. The advantage of using Bloom filters [27] is that they are space efficient, i.e., with a small space, one can index a large number of data. However, it is possible that a Bloom filter gives a *false positive* answer, i.e., the Bloom filter wrongly returns a positive answer in response to a question asking the membership of a data item. An important feature of a Bloom filter-based index is that it minimizes the maintenance overhead, making it suitable for highly dynamic environments.

In [11], each peer replicates d copies of its Bloom filter and distribute them to its neighbors. Then, peers periodically exchange with each other the Bloom filters they have, so as to widely disseminate them in the P2P network. Each Bloom filter is associated a tag TTL that records the time up to which a Bloom filter is valid. When the time expires, the peer that holds the copy should check with the owner of the copy to obtain a new version. The approach in [7] was previously introduced in Section 4.0.4 where a peer has two types of neighbors in the P2P overlay: interest-based and proximity-based. Thus, each peer stores the Bloom filters of both types of neighbors. The advantage of this approach over the previous indexing schemes is that it attemps to achieve a locality-aware routing.

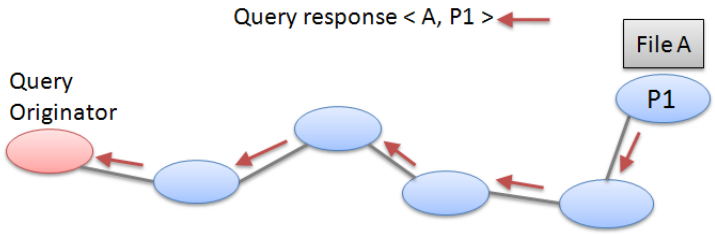
Index Caching. The basic concept is to cache query responses in the form of indexes, on their way back to the originator. Recall that a query response contains the file indetifier (e.g., filename) and the address of the provider peer that has a copy of the file. The advantage of index caching is that it does not incur additional overhead to create and update the indexes, as they exploit passing-by query responses.

Let us briefly review the different approaches of index caching. Centralized caching [64] at the gateway of an organization does not leverage node resources and is likely to produce bottlenecks. *Distributed index caching* is illustrated in Figure 16, where a query requesting file A has reached peer P1 that can provide a copy of A (see Figure 16a). As normally done in unstructured systems, a query response that contains the filename of A and the address of P1 is sent back to the query originator. Forwarding peers cache the query response as an index for file A and thus can respond to eventual queries requesting file A. *Uniform index caching* [84] consists that each peer caches all passing-by query responses, which results in large amount of duplicated and redundant cached among neighboring nodes (see Figure16b). *Selective index caching* addresses the problem of redundancy by selectively caching file indexes and accordingly routing queries. However, none of the existing solutions addresses locality-awareness in file transfers. Next, we describe a typical example of selective index caching, i.e., *DiCAS* [95]. Then we present *Locaware* [25, 21] which introduces locality-awareness in index caching.

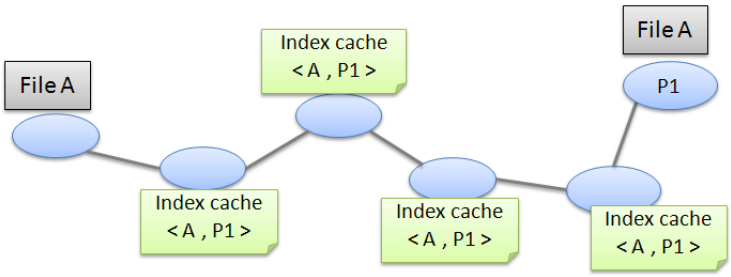
DiCAS. Peers are randomly assigned to M groups, with each group being identified by an ID noted Gi. Group IDs are used to restrict index caching in some peers along the query reverse path, in order to avoid redundant indexes among neighbors. Hence, a query response is only cached in peers whose Gi matches the filename in the query response, i.e., Gi = hash(f) mod M (see Figure17a). Furthermore, Group IDs help searching for file indexes by routing a query towards peers that are likely to have indexes satisfying the query. To forward a query, a peer selects the neighbors whose Gi matches the string of keywords in the query (see Figure17b). When no such neighbors are found, the query is sent to a highly connected neighbor.

This search is specifically tailored for exact-match queries. Therefore, DiCAS is not adapted for keyword searches which are the most common in the context of P2P file sharing. To illustrate this problem, consider a user looking for a file with name $f = key_1 + key_2 + ... + key_n$. Commonly, the user will employ a query with string of keywords $q = key_1 + key_m + key_n$. Based on DiCAS predefined hashing, the file index is cached in peers with Gi = hash(f) mod M, while the query is forwarded to peers with Gi' = hash(q) mod M. Obviously, this approach may mislead the query by redirecting it to peers that have no indexes for the target file. This results in more flooding overhead and higher response time.

One alternative is to cache file indexes based on the hashing of the query string of keywords (i.e, q). Since multiple combinations of keywords can map to the same filename, it brings back the problem of wide duplication and reduces the efficiency of indexes.



(a) Query response on its way back to the originator.



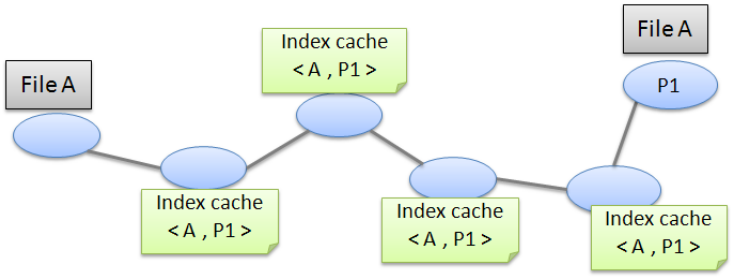
(b) Query response cached by all forwarding peers.

Figure 16: Uniform index caching.

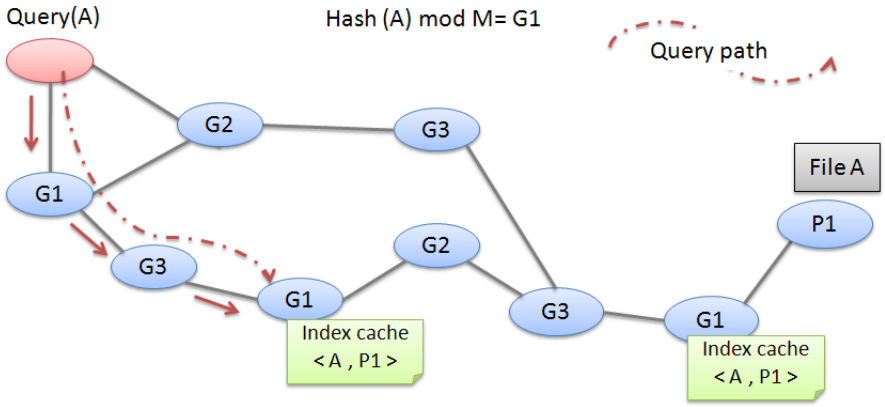
Locaware. Locaware [25, 21] borrows the selective index caching technique of DiCAS which is based on filename hashing and group IDs. To support keyword queries, each peer maintains a Bloom filter that represents the set of keywords of all cached filenames in its index cache. Whenever the peer caches a file index, it inserts each keyword of the cached filename as an element of its Bloom filter. A Bloom filter BF *matches* a query $q = \{key_1, ..., key_n\}$ if $\forall key_i \in q;\ key_i \in BF$. Each peer replicates its Bloom filter and sends a copy to each one of its direct neighbors. A peer delays the propagation of its BF updates to its neighbors until the rate of new changes in its index cache reaches a threshold. As a result, a peer can query its neighbors' Bloom filters to selectively route a query.

To enable locality-awareness, Locaware exploits natural file replication, based on the fact that a peer which has recently requested a file F is likely to have it and can thereby serve subsequent requests for F. Localities are modeled via the binning technique [70] where each possible bin is associated a locality Id noted $locId$. Upon joining the network, each peer computes its own $locId$.

The index cache of a peer may hold for a cached filename, several provider addresses and their $locId$s (see Figure 18). To achieve this, a query response should contain both the address and the $locId$ of the file provider. Additionally, it includes the address and the $locId$ of the query originator, which will be considered as a new provider by peers intercepting the responses. In other terms, a peer that is forwarding a query response checks if the vehiculated



(a) Query response selectively cached by forwarding peers.



(b) Query selectively routed to neighbors.

Figure 17: Selective index caching: DiCAS with M = 3 groups G1,G2,G3.

filename matches its group ID. If so, the peer extracts from the query response the address information about both the provider and the query originator to cache them. This is illustrated in Figure 18 where peers are assigned to three groups G1, G2 and G3 (i.e., $M = 3$). P2 requests the file whose name F matches G1; its query has reached a peer from G1 that has an index for F related to provider P1. The latter peer generates a query response $< F, P1, locId = 1 >$ and caches a new index for F related to the evenual provider P2. Then, peers of G1 forwarding the query response cache two indexes for F, one related to P1 and another to P2.

5.2.3 Discussion

In summary, P2P file sharing is a highly popular application that tolerates some performance limitation and prefers unstructured overlays. However, there is a growing concern regarding its network costs since the P2P traffic overwhelms

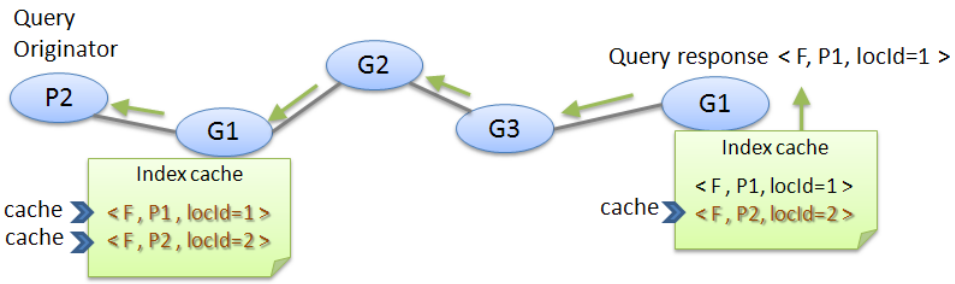


Figure 18: Locaware. Caching indexes of filename F; hash(F) mod M = G1.

the Web traffic as a leading consumer of Internet bandwidth. Two main reasons are behind this problem.

First, searching for files is inefficient, generating large amounts of redundant messages. Indexing can help improve search efficiency as it provides information useful for query routing. However, it might imply considerable overhead for the creation and update of indexes. Furthermore, it is highly required from file sharing applications to support keyword lookup. Thus, indexing should not hinder this feature.

Second, the large files are transferred over long network distances, thus overloading the underlying network. Locality-awareness seem to be the best solution for this issue, redirecting queries to close-by files.

On the other hand, there are several inherent properties of P2P file sharing that have not been fully exploited and could be leveraged to attenuate the P2P traffic problem. The most important ones are the temporal locality of queries and the natural replication of files. For instance, index caching leverage temporal locality as it keeps query responses for later queries in order to improve search efficiency.

To conclude, Table 3 lists indexing approaches and checks whether or not they answer the different challenges. As an example, the first two approaches do not address the overhead related to their index creation and maintenance. On the same matter, index caching like DiCAS and Locaware implicitly limits the overhead since it dynamically stores and evicts indexes as query responses pass by. Also, the usage of Bloom filters can achieve, at the same time, maintenance and storage efficiency. Another observation is that most of the approaches do not incorporate locality-awareness in their indexing scheme and thus fail in redirecting queries to nearby locations for short data tranfers.

5.3 P2P CDN

Several P2P approaches have been proposed to distribute web content over P2P infrastructures in order to relieve the original web servers . As previously discussed, they can greatly optimize their performance if they take into account recent P2P trends while meeting their challenges (cf. Section 4). We classify existing approaches into three main categories: hybrid, unstructured and DHT-

Indexing approach	Overhead efficiency	Keyword lookup	Locality-awareness
Routing indices	no	yes	no
Local indices	no	yes	no
Intelligent BFS	no	yes	no
Bloom filter indices	yes	yes	yes
DiCAS	yes	no	no
Locaware	yes	yes	yes

Table 3: File indexing approaches

based. We also distinguish the currently deployed P2P CDNs. First of all, let us give an overview of caching and replication mechanisms. Then we survey the existing P2P CDNs and investigate if they meet the requirements and leverage the recent trends.

5.3.1 Insights into Caching and Replication

In the context of content distribution, content replication is commonly used to improve content availability and enhance performance. More particularly, P2P systems can significantly benefit from replication given the high levels of dynamicty and churn. For instance, if one peer is unavailable, its objects can still be retrieved from the other peers that hold replicas. According to [1], content replication in P2P systems can be categorized as follows.

Passive Replication. It refers to the replication of content that occurs naturally in P2P systems as peers request and download content. This technique perfectly complies with the autonomy of peers.

Active (or Proactive) Replication. This technique consists in monitoring traffic and requests, and accordingly creating replicas of content objects to accommodate future demand.

To improve object availability and at the same time avoid hotspots, most DHT-based systems replicate popular objects and maps the replicas to multiple peers. Generally, this can be done via two techniques. The first one [69] uses several hash functions to map the object to several keys and thereby store copies at several peers. The second technique consists in replicating the object in a number of peers whose IDs match most closely the key (or in other terms, in the logical neighborhood of the peer whose ID is the closest to the key). The latter technique is commonly used in several systems such as [77, 17].

The study in [13] evaluate three different strategies for replication in an unstructured system. The *uniform strategy* creates a fixed number of copies when the object first enters the system. The *proportional strategy* creates a fixed number of copies every time the object is queried. In the *square-root replication strategy*, the ratio of allocations is the square root of the ratio of query rates. To implement these strategies, the object can be replicated either randomly or at peers along the path from the requester peer to the provider peer. However, it is not clear how the strategies can be achieved in a distributed way (e.g., how to monitor query rate under P2P dynamicity). Further, such proactive replication

is not feasible in systems that wish to respect peer autonomy; they may not want to store an object at peers that have not requested it.

Along with replication, there is the classical issue of maintaining consistency between replicas in case of content updates. In this paper, we do not discuss this issue, however good pointers can be found in our previous work [57, 20].

Caching The key idea is to cache copies of content as it passes through peers in the network and manage them according to cache replacement policies. In Freenet [12] for instance, when a search request succeeds in locating an object, the object is transferred through the network node-by-node back to the query originator. In the process, copies of the object are cached by all intermediate nodes.

To shorten query routes and thus reduce search latencies, DHT-based approaches like [77, 17] cache additional copies of the objects along the lookup path towards the peers storing these objects.

5.3.2 Deployed Systems

To the best of our knowledge, the P2P CDNs that are currently available for public use mainly comprise CoralCDN [29], CoDeeN [62] and CobWeb [83]. These systems are deployed over PlanetLab which provides a relatively trusted environment consisting of nodes donated largely by the research community. Basically, they rely on a network of cooperative proxy servers that distribute web content and handle related queries. Such systems cannot be categorized as pure P2P solutions because they are using dedicated servers rather than exploiting client resources. The only P2P characteristic exhibited by these systems is the absence of centralized administartion. We examine one typical example of these systems, CoralCDN.

CoralCDN [29]. CoralCDN relies on a hierarchy of tree-based overlays that cluster nearby nodes. Each level of the hierarchy consists of several overlays, and each overlay consists of the set of nodes whose average pair-wise RTTs are below the threshold defined by this level. A node is member of one overlay at each hierarchy level and detains the same node ID in all overlays to which it belongs. Figure 19 illustrates a three-level hierarchy with RTT thresholds of ∞, 60 msec, and 20 msec for level 0, 1, and 2 respectively. It focuses on Node R and only shows the three overlays to which R belongs at each level. R is physically the closest to C_2 among the nodes (C_0, C_1, C_2, C_3) because R and C_2 share the highest-level overlay.

Each overlay is structured according to a tree topology. A key is mapped to several nodes whose IDs are numerically close to the key, in order to avoid hot spots due to popular objects. A node stores pointers related to the object whose key is mapped to its node ID. In Figure 19, Node R has the same node ID in all its overlays; we can view a node as projecting its presence to the same logical location in each of its overlays.

Based on this indexing infrastructure, CoralCDN allows to locate web object copies hosted by nearby proxies of CoralCDN: the proxies will be represented by the nodes of the hierarchy. Based on its RTT measurements, a client is redirected via the DNS services to a nearby CoralCDN proxy which eventually provides her the requested object. If not cached locally, the proxy can perform a

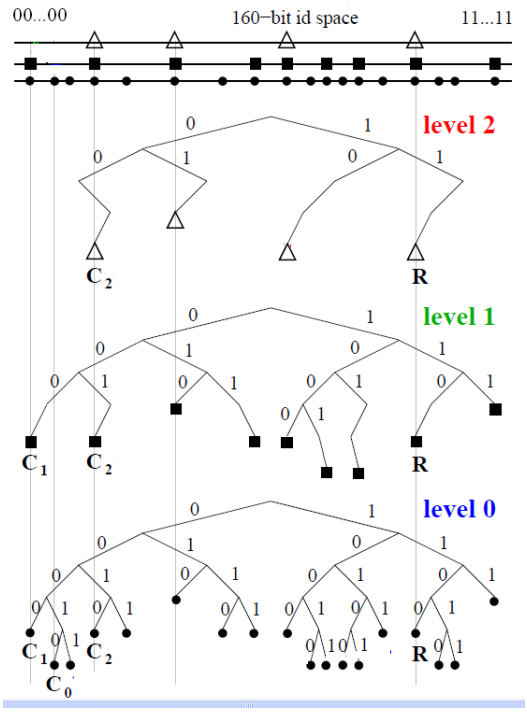


Figure 19: CoralCDN hierarchy of key-based overlays [29].

key-based routing throughout its overlays in order to find a pointer to a remote copy of the object; it starts at the highest-level overlay of the proxy to benefit from network locality then progresses down the hierarchy. Once the object is fetched and locally cached, the proxy inserts pointers to itself wrt. the object in the different overlays to which belongs this proxy. To handle dynamicity, pointers are associated with ttl values and are periodically refreshed by their referenced proxy.

5.3.3 Centralized Approaches

The first category of approaches [79, 61] relies on the web-server that centralizes and manages the directory information. Basically, the server maintains a directory of peers to which its objects have been transferred in the past and manages the redirection of queries. When a client requests an object, the server returns several peers from the redirection directory. The client first tries to retrieve the object from one of those peers. If this fails, the object is directly served by the server.

To minimize redirection failures in a P2P dynamic environment, OLP [79] tries to predict the object lifetime and accordingly selects the peer to which the query should be redirected. However, redirection in OLP does not consider locality-awareness when providing clients with object locations. CoopNet [61] tries to incorporate locality-awareness as the web-server sends to the requester

client a list of nearby peers providing the requested object. To limit the server redirection, a client connects to the peers provided by the web-server and forms a small network with them. However, there is no well-defined search algorithms within these networks. Moreover, CoopNet does not deal with dynamic aspects because the web-server cannot detect which peers in its directory have failed or discarded their cached objects.

Centralized approaches lack robustness, because whenever the web-server fails, its content is no longer accessible in spite of available peers with cached copies. As with the traditional server/client model, the server is still a single point of failure. Scaling such systems requires replacing the web server with a more powerful one, to be able to redirect the queries of a large audience.

5.3.4 Unstructured Approaches

The second category of approaches uses unstructured overlays for their flexibility and inherent robustness. Two representative systems are Proofs and BuddyWeb.

Proofs. Proofs [88] uses an ustructured overlay in which peers continuously exchange neighbors among each other. This provides each peer with a random view of the system for each search operation. Peers keep their requested objects and can then provide them to other participants. To locate one of the object replicas, a query is flooded to a random subset of neighbors with a *fixed time-to-live* (TTL) i.e., the max number of hops. The continuous randomization of the overlay has the benefit of improving the network fault-tolerance and tends to uniformly distribute the load over peers. However, the blind searches for not not-so popular objects induce heavy traffic overheads and high latencies. Moreover, Proofs does not leverage new trends, and most importantly locality-awareness which is useful to forward queries to close results.

BuddyWeb. BuddyWeb [97] also uses an unstructured network and blind search to access objects. However, it relies on central servers to provide each newly joining peer with neighbors that share interest similarities with the peer. Therefore, this interest-based scheme does not meet Challenge 1 as it greatly depends on central servers to gather, manage and provide all the information. These servers can present single points of failures (i.e., SPOF), which makes BuddyWeb vulnerable and hinders its scalability. Similarly to Proofs, Buddy-Web does not take into account locality-awareness.

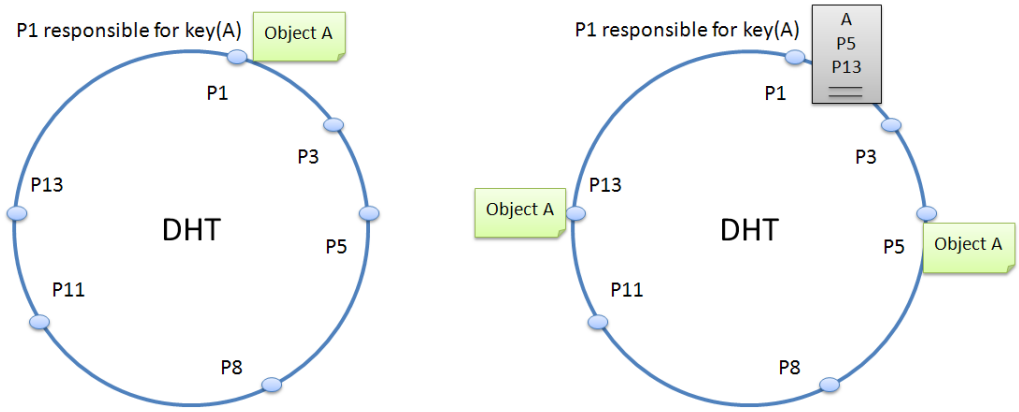
5.3.5 Structured Approaches

Now, we examine existing approaches that rely on structured overlays in order to benefit from their efficient lookup. We examine Squirrel [38], PoPCache [68] and Backslash [86]. These approaches adopt similar strategies. We have identified two types of strategies, *home-based* and *directory-based*. We also discuss a work that proposes a different approach using a novel DHT, called *Kache*.

Home-Based Strategy. It places objects at peers with ID numerically closest to the hash of the URL of the object without any locality or interest considerations (see Figure20a). Queries find the peer that has the object by navigating

through the DHT. To deal with highly popular objects, objects may be progressively replicated along neighbors as the number of requests increases. This is achieved by further forcing peers to store arbitrary content.

Directory-Based Strategy. The second type of strategy stores at the peer identified by the hash of the object's URL a small directory of pointers to recent downloaders of the object (see Figure20b). A query first navigates through the DHT and then receives a pointer to a peer that potentially has the object. Approaches adopting this strategy may be vulnerable to high churn because the directory information is abruptly lost at the failure of its storing peer.



(a) Home model. (b) Directory model.

Figure 20: Squirrel-like strategies in a P2P CDN.

We refer to these DHT-based systems as Squirrel-like systems because Squirrel has both DHT strategies. In general, such systems are self-scalable because of the DHT load balancing mechanism and the replication in case of hot spots. However, there are two main drawbacks in the query routing with repsect to the stringent requirement of CDNs on short latencies. First, each query has to navigate through the whole DHT, which implies several routing hops. This can be acceptable in corporate LAN type environments, such that the latency of the network links are a magnitude smaller than the latency of the server. Otherwise, the server will be much faster. Second, unless using a locality-aware overlay combined with proactive replication, the query is served from a random physical location. To conclude, the aforementioned approaches do not exploit recent P2P trends for performance improvement.

Kache. Kache [52] relies on a new form of DHT that increases robustness to churn by increasing memory usage and communication overhead. Basically, peers are organized using a hash function into $\sqrt{N}$ groups where N = total number of peers. This is shown in Figure 21 with focus on the peer with ID=110 from group 0. The peer maintains (a) a view of its own group (i.e.,

peers 30 and 160), and (b) for each foreign group, a small (constant-sized) set of contact peers lying in it (i.e., peer 432). Each entry (group view or contact) carries additional fields such as RTT estimates. Peer 110 also stores directory

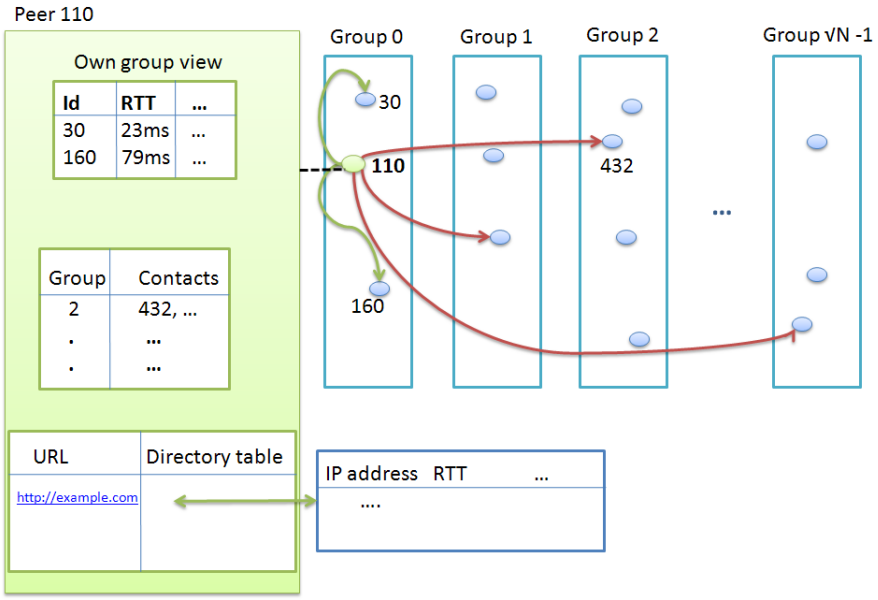


Figure 21: A Kache system with peers distributed across $\sqrt{N}$ groups, and soft state at a typical peer [52].

information related to each single object that is cached in the system and whose URL maps to group 0 by means of hashing. For each such object o, peer 110 has a *directory table* that contains the IP addresses of a bounded set of peers holding a copy of o.

When a peer p downloads a copy of the object o, it creates a directory entry $< o, p >$ and communicates it to its contacts c that belong to o's group. When the directory table of c is full, c performs RTT measurements to keep the directory entries that refer to the closest peers and discard the other entries. Each peer gossips within its group to replicate and spread directory entries; it selects close-by peers from its view to exchange gossip messages. Obviously, peers gossiping and replicating directory entries are not necessarily interested in this information. Furthermore, since directory information is highly replicated, aggressive updates are required when referenced peers discard their content or leave the network.

Kache is robust against failures, because all peers in the same group store pointers of all the objects mapped onto the group. Moreover, locality-awareness is incorporated through the RTT-based routing tables. Lookups are bounded by $O(1)$, thus scaling does not influence lookup time. However, the resources necessary to maintain routing information increases as the number of peers increases.

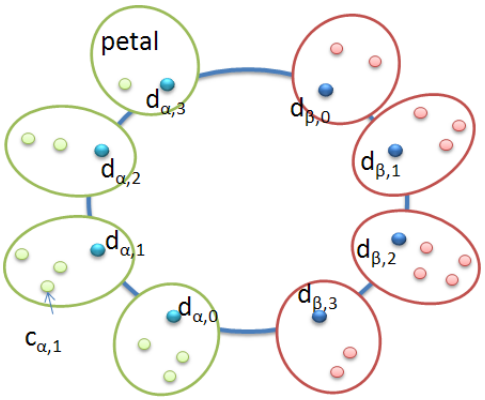


Figure 22: Flower-CDN architecture with websites α and β and four localities.

Flower-CDN. Flower-CDN [22, 24] distributes the popular content of any under-provisioned website by stricly relying on the community of users interested in its content. To this end, it takes into account the interests and localities of users, and accordinly organizes peers and serves queries. Locality-awareness is implemented via the binning technique [70] where each bin identifies a locality *loc*. Moreover, Flower-CDN adopts a hybrid architecture that combines the strengths of structured and unstructured overlays.

Figure 22 illustrates the architecture of Flower-CDN. Participant peers belonging to the same locality *loc* and interested in the same website *ws* build together an unstructured overlay noted $petal(ws, loc)$. These peers, called *content peers* (i.e., $c_{ws,loc}$), cache and manage content of *ws*. Within a petal, content peers use Bloom filters and gossip protocols to exchange information about their contacts and their cached content.

Flower-CDN charges one peer of each $petal(ws, loc)$, the role of a *directory peer* (i.e., $d_{ws,loc}$): $d_{ws,loc}$ knows about all content peers $c_{ws,loc}$ and keeps information about their stored content. Directory peers are also embedded in *D-ring*, a DHT-structured overlay that is adapted to reflect interests and localities.

Instead of querying the server *ws*, a new client located in *loc*, submits its query to D-ring and gets directed to the directory peer in charge of *ws* in *loc* i.e., $d_{ws,loc}$. Then $d_{ws,loc}$ tries to resolve the query while relying on its petal or some neighboring petals related to *ws*. Eventually the query is redirected to a content peer $c_{ws,loc}$ that holds the requested object and hence serves the query. Then the client joins $petal(ws, loc)$ as a content peer $c_{ws,loc}$. The two-layered infrastructure consisting of D-ring and the petals serves as a locality-aware query routing and serving. D-ring ensures a reliable access for new clients, whereas the petals allow them to subsequently perform locality-aware searches. Thus, most of the query routing takes place within a locality-based cluster leading to short response times and local data transfer.

To achieve high scalability, PetalUp-CDN [23] has been proposed to extend Flower-CDN. D-ring progressively expands in order to manage larger petals while avoiding overload situations. Basically the number of directory peers in

a petal increases with the number of content peers. Multiple directory peers share the management of a given petal, mainly in indexing the petal's content and servicing new clients. Moreover, PetalUp-CDN deals with reverse contexts where peers progressively depopulate the petals, by dynamically shrinkng D-ring.

Robustness is ensured via gossip-based protocols that continuously operate in the background to monitor peers and update the system according to churn and dynamicity. Flower-CDN remediates to the overhead of gossip protocols in terms of messages and delay by confining them in localities such that gossip exchanges only engage close-by peers. Further D-ring attenuates the DHT problem with respect to the maintenance cost. Since only a selective set of participants take part of D-ring, the number of maintenance messages is significantly reduced.

5.3.6 Discussion

Table4 summarizes the performance behavior of the P2P-CDN approaches previously described. Obviously, none of them fully satisfy the requirements that we have identified along this paper. An important observation is that most of the approaches do not focus on scalability, and often target small local networks.

In CoralCDN, users are not involved in the P2P network: they use the P2P CDN but do not contribute any resources to it. An increase of the number of users requires more investment by adding proxy caches to the CoralCDN. OLP is unsuitable for P2P systems as it is not scalable (i.e., bottlenecks) nor robust to churn (i.e., SPOF) due to its centralized nature, and it does not consider locality-awareness. CoopNet has similar limitations, except that it supports locality-aware redirection of queries. Proof derives its robustness from the randomness of unstructured overlays, but in return suffers from their scalability issues due to flooding overhead and lacks locality-awareness. BuddyWeb does not cope with dynamic and large-scale participation of peers because its construction mechanism is centralized, and thus is not adapted for real P2P environments. Kache addresses most of the requirements, and most importantly achieves robustness by replicating and gossiping indexing information. However, Kache scalability comes at the cost of a significant storage overhead on every peer. Squirrel-like systems that adopt the directory strategy do not provide robustness as the performance of query handling is directly affected by peer failures. In comparison, Squirrel-like systems with the home strategy rely on DHT robustness which incurs high costs and breaks the autonomy of peers. In addition, all Squirrel-like systems do not specifically incorporate locality-awareness which is a major requirement of P2P-CDN. Finally, Flower-CDN succeeds in achieving all the requirements.

6 Conclusion

The objective of this report was to provide a consice, yet comprehensive study of P2P content distribution. For this, we reviewed the state-of-the-art for traditional and P2P content distribution in order to identify the shortcomings and highlight the challenges.

SYSTEM	OVERLAY	ROBUSTNESS	SCALABILITY	LOCALITY	AUTONOMY
CoralCDN	hierarchy of proxies	yes	more proxy investment	yes	-
OLP	centralized	SPOF	server bottleneck	no	yes
CoopNet	centralized	SPOF	server bottleneck	no	yes
Proof	unstructured	randomness	flooding overhead	no	yes
BuddyWeb	unstructured	SPOF	server bottleneck	no	yes
Kache	gossip/DHT	replication	overhead	yes	yes
Squirrel/directory	structured	directory loss	yes	no	yes
Squirrel/home	structured	DHT robustness	yes	no	no
Flower-CDN	hybrid	yes	yes	yes	yes

Table 4: Summary of P2P-CDNs

First, we identified the traditional requirements for CDNs which are performance, scalability and reliability, and we discussed the mechanisms needed to fulfill each requirement. We also shed light on CDN open issues, mainly in terms of scalabilty and its significant costs. We focused on the potential savings and benefits in using P2P technology as a cheap and efficient alternative for commercial CDNs.

Second, we explored P2P systems from the perspective of content sharing and shed light on the design requirements that are crucial to make efficient use of P2P self-scalability. The main relevant requirements are autonomy, expressiveness, efficiency, quality of service, and robustness.

Third, we presented the recent P2P trends that can improve the performance of P2P content distribution but incur additional challenges. The trends that we identified are locality-aware and interest-aware overlay matching, gossip usage and overlay combination. The challenges are to keep the solutions simple, avoid centralized management and large overheads, operate fast and dynamically adapt to changes and massive scales. Along these lines, matching the overlay with a locality- or interest-aware scheme could bring great benefits to the P2P system in terms of efficiency and quality of service. Another recent trend is the combination of different overlays and schemes, which can reveal very challenging. In particular, the maintenance of several overlays should not overwhelm the P2P system. Gossip protocols can serve as potentially effective means to achieve these new trends as it provides simplicity, decentralization and high robustness to churn. However, gossip should be properly designed and tuned to avoid significant delays and message overheads.

Finally, we focused on the two P2P applications that derive from content distribution, P2P file sharing and P2P CDN. We investigated both fields and reviewed existing approaches. In the context of file sharing, there is a growing concern about the network costs since the P2P traffic is the leading consumer of Internet bandwidth, mainly due to search inefficiency and long file transfer. While the top priority is to exploit locality-awareness in order to serve queries from close-by locations, most existing works do not address this issue. Regarding P2P CDN, they have stringent performance requirements that are quire different to what is expected from a file-sharing system. They should be highly robust, efficient and scalable, while taking into account the autonomy of peers. Existing P2P CDNs do not answer all the important requirements. Most of them are not designed to achieve high scalability as they target small scales.

References

[1] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[2] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 205–217, 2002.

[3] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[4] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[5] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content Delivery Networks*. LNEE. Springer, 2008.

[6] Ramón Cáceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: the devil is in the details. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):11–15, 1998.

[7] Hailong Cai and Jun Wang. Foreseer: A novel, locality-aware peer-to-peer system architecture for keyword searches. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 38–58, 2004.

[8] Bengt Carlsson and Rune Gustavsson. The rise and fall of napster - an evolutionary approach. In *Proceedings of the 6th International Computer Science Conference on Active Media Technology (AMT)*, volume 2252 of *LNCS*, pages 347–354. Springer, 2001.

[9] Miguel Castro, Manuel Costa, and Antony I. T. Rowstron. Should we build Gnutella on a structured overlay? *ACM SIGCOMM Computer Communication Review*, 34(1):131–136, 2004.

[10] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 407–418, 2003.

[11] An-Hsun Cheng and Yuh-Jzer Joung. Probabilistic file indexing and searching in unstructured peer-to-peer networks. *Computer Networks*, 50(1):106–127, 2006.

[12] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[13] Edith Cohen and Scott Shenker. Replication strategies in unstructured P2P networks. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures,and Protocols for Computer Communications (SIGCOMM)*, pages 177–190, 2002.

[14] Ian Cooper, Ingrid Melve, and Gary Tomlinson. Internet Web replication and caching taxonomy. Internet Engineering Task Force RFC 3040, 2001. `www.ietf.org/rfc/rfc3040.txt`.

[15] Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 23–, 2002.

[16] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems. In *Proceedings of the 3rd International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, volume 3601 of *LNCS*, pages 1–13. Springer, 2004.

[17] Frank Dabek, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 202–215, 2001.

[18] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *Proceedings of the 9th International Conference on Database Theory (ICDT)*, volume 2572 of *LNCS*, pages 1–15. Springer, 2002.

[19] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[20] Manal El Dick, Vidal Martins, and Esther Pacitti. A topology-aware approach for distributed data reconciliation in P2P networks. In *Proceedings of the 13th International Conference on Parallel and Distributed Computing (Euro-Par)*, volume 4641 of *LNCS*, pages 318–327. Springer, 2007.

[21] Manal El Dick and Esther Pacitti. Locaware: Index caching in unstructured P2P-file sharing systems. In *Proceedings of the 2nd ACM International Workshop on Data Management in Peer-to-peer systems (DAMAP)*, page 3, 2009.

[22] Manal El Dick, Esther Pacitti, and Bettina Kemme. Flower-cdn: a hybrid P2P overlay for efficient query processing in CDN. In *Proceedings of the 12th ACM International Conference on Extending Database Technology (EDBT)*, pages 427–438, 2009.

[23] Manal El Dick, Esther Pacitti, and Bettina Kemme. A highly robust P2P-CDN under large-scale and dynamic participation. In *Proceedings of the 1st International Conference on Advances in P2P Systems (AP2PS)*, pages 180–185, 2009.

[24] Manal El Dick, Esther Pacitti, and Bettina Kemme. Un réseau pair-à-pair de distribution de contenu exploitant les intérêts et les localités des pairs. In *Actes des23èmes Journées Bases de Données Avancées, (BDA) (Informal Proceedings)*, pages 407–388, 2009.

[25] Manal El Dick, Esther Pacitti, and Patrick Valduriez. Location-aware index caching and searching for P2P systems. In *Proceedings of the 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2007.

[26] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulieacute. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[27] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 291–293, 1998.

[28] Fabrice Le Fessant, Sidath B. Handurukande, Anne-Marie Kermarrec, and Laurent Massoulié. Clustering in P2P file sharing workloads. In *Proceedings of the 3rd International Workshop on P2P Systems (IPTPS)*, volume 3279 of *LNCS*, pages 217–226. Springer, 2004.

[29] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 239–252, 2004.

[30] Gnutella protocol specification. `http://wiki.limewire.org/index.php?title=GDF`, 2005.

[31] P. Krishna Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proceedings of the 19th ACM Symposium on Operating systems principles (SOSP)*, pages 314–329, 2003.

[32] P. Krishna Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures,and Protocols for Computer Communications (SIGCOMM)*, pages 381–394, 2003.

[33] Sidath B. Handurukande, Anne-Marie Kermarrec, Fabrice Le Fessant, and Laurent Massoulié. Exploiting semantic clustering in the eDonkey P2P network. In *Proceedings of the 11th ACM SIGOPS European Workshop*, page 20, 2004.

[34] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in DHT-based peer-to-peer networks. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, LNCS, pages 242–259. Springer, 2002.

[35] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB)*, pages 321–332, 2003.

[36] AccuStream iMedia Research. CDN market growth 2006 - 2009, 2008. `www.researchandmarkets.com`.

[37] ITU World Telecommunication/ICT Indicators Database. International Telecommunication Union - Development Sector, 2007. `http://www.itu.int/ITU-D/ict/statistics/ict/index.html`.

[38] Sitaram Iyer, Antony I. T. Rowstron, and Peter Druschel. Squirrel: a decentralized P2P web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 213–222, 2002.

[39] Márk Jelasity and Özalp Babaoglu. T-Man: Gossip-based overlay topology management. In *Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems (ESOA)*, volume 3910 of *LNCS*, pages 1–15. Springer, 2005.

[40] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, volume 3231 of *LNCS*, pages 79–98. Springer, 2004.

[41] Mihajlo A. Jovanovic. Modelling large-scale peer-to-peer networks and a case study of Gnutella. Master's thesis, ECECS Department, University of Cincinnati, 2000.

[42] Mihajlo A. Jovanovic, Fred S. Annexstein, and Kenneth A. Berman. Scalability issues in large peer-to-peer networks: a case study of Gnutella. Technical report, ECECS Department, University of Cincinnati, 2001.

[43] Vana Kalogeraki, Dimitrios Gunopulos, and Demetrios Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 300–307, 2002.

[44] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC)*, pages 63–76, 2005.

[45] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th ACM Symposium on the Theory of Computing (STOC)*, pages 654–663, 1997.

[46] Magnus Karlsson and Mallik Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW)*, pages 117–128, 2002.

[47] Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *ACM SIGOPS Operating Systems Review*, 41(5):2–7, 2007.

[48] Balachander Krishnamurthy, Jia Wang, and Yinglian Xie. Early measurements of a cluster-based architecture for P2P systems. In *Proceedings of the 1st ACM SIGCOMM Internet Measurement Workshop*, pages 105–109, 2001.

[49] John Kubiatowicz, David Bindel, Yan Chen, Steven E. Czerwinski, Patrick R. Eaton, Dennis Geels, Ramakrishna Gummadi, Sean C. Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Y. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 190–201, 2000.

[50] Nathaniel Leibowitz, Aviv Bergman, Roy Ben-shaul, and Aviv Shavit. Are file swapping networks cacheable? characterizing P2P traffic. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW)*, 2002.

[51] Jian Lianga, Rakesh Kumarb, and Keith W. Ross. The FastTrack overlay: A measurement study. *Computer Networks Journal*, 50(6):842–858, 2006.

[52] Prakash Linga, Indranil Gupta, and Ken Birman. A churn-resistant P2P web caching system. In *Proceedings of the ACM Workshop on Survivable and Self-Regenerative Systems (SSRS)*, pages 1–10, 2003.

[53] Yunhao Liu, Li Xiao, Xiaomei Liu, Lionel M. Ni, and Xiaodong Zhang. Location awareness in unstructured P2P systems. *IEEE Transaction on Parallel and Distributed Systems*, 16(2):163–174, 2005.

[54] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th ACM International Conference on Supercomputing (ICS)*, pages 84–95, 2002.

[55] Balasubramaneyam Maniymaran, Marin Bertier, and Anne-Marie Kermarrec. Build one, get one free: Leveraging the coexistence of multiple P2P overlay networks. In *Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS)*, page 33, 2007.

[56] Evangelos P. Markatos. Tracing a large-scale peer-to-peer system: An hour in the life of Gnutella. In *Proceedings of 2nd the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 65–74, 2002.

[57] Vidal Martins, Esther Pacitti, Manal El Dick, and Ricardo Jiménez-Peris. Scalable and topology-aware reconciliation on P2P networks. *Journal of Distributed and Parallel Databases*, 24(1-3):1–43, 2008.

[58] C. Mohan. Caching technologies for Web applications. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, page 726, 2001.

[59] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: a P2P networking infrastructure based on rdf. In *Proceedings of the 11th ACM International Conference on World Wide Web (WWW)*, pages 604–615, 2002.

[60] Nikos Ntarmos and Peter Triantafillou. Aesop: altruism-endowed self-organizing peers. In *Proceedings of the 2nd International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*, volume 3367/2005 of *LNCS*, pages 151–165. Springer, 2004.

[61] Venkata N. Padmanabhan and Kunwadee Sripanidkulchai. The case for cooperative networking. In *Proceedings of the 1st International Workshop on P2P Systems (IPTPS)*, volume 2429 of *LNCS*, pages 178–190. Springer, 2002.

[62] Vivek S. Pai, Limin Wang, KyoungSoo Park, Ruoming Pang, and Larry Peterson. The dark side of the Web: an open proxy's view. *ACM SIGCOMM Computer Communication Review*, 34(1):57–62, 2004.

[63] George Pallis and Athena Vakali. Insight and perspectives for content delivery networks. *Communications of the ACM*, 49(1):101–106, 2006.

[64] Sunil Patro and Y. Charlie Hu. Transparent query caching in peer-to-peer overlay networks. In *Proceedings of the 17th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, page 32a, 2003.

[65] Gang Peng. CDN: Content distribution network. Technical report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, 2003.

[66] Johan A. Pouwelse, Pawel Garbacki, Dick H. J. Epema, and Henk J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *Proceedings of the 4th International Workshop on P2P Systems (IPTPS)*, volume 3640 of *LNCS*, pages 205–216. Springer, 2005.

[67] Yi Qiao and Fabián E. Bustamante. Structured and unstructured overlays under the microscope: a measurement-based view of two P2P systems that people use. In *Proceedings of the USENIX Annual Technical Conference (ATEC)*, pages 341–355, 2006.

[68] Weixiong Rao, Lei Chen 0002, Ada Wai-Chee Fu, and Yingyi Bu. Optimal proactive caching in P2P network: analysis and application. In *Proceedings of the 6th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 663–672, 2007.

[69] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 161–172, 2001.

[70] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM)*, pages 1190–1199, 2002.

[71] Sylvia Ratnasamy, Ion Stoica, and Scott Shenker. Routing algorithms for DHTs: Some open questions. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *LNCS*, pages 45–52. Springer, 2002.

[72] Robbert Van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. Technical report TR98-1687, Cornell University, 1998.

[73] Sean C. Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference, General Track*, pages 127–140, 2004.

[74] Matei Ripeanu, Ian T. Foster, and Adriana Iamnitchi. Mapping the Gnutella network: Properties of large-scale P2P systems and implications for system design. *IEEE Internet Computing*, 6(1):50–57, 2002.

[75] Jordan Ritter. Why Gnutella can't scale, no, really. `http://www.darkridge.com/jpr5/doc/gnutella.html`, 2001.

[76] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale P2P systems. In *Proceedings of the 2nd ACM/IFIP International Conference on Middleware*, volume 2218 of *LNCS*, pages 329–350. Springer, 2001.

[77] Antony I. T. Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, 2001.

[78] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC)*, volume 2233 of *LNCS*, pages 30–43. Springer, 2001.

[79] Young-Suk Ryu and Sung-Bong Yang. An effective P2P web caching system under dynamic participation of peers. *IEICE Transactions*, 88-B(4):1476–1483, 2005.

[80] Stefan Saroiu, P. Krishna Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of Internet content delivery systems. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 315–327, 2002.

[81] Haiying Shen and Cheng-Zhong Xu. Hash-based proximity clustering for efficient load balancing in heterogeneous DHT networks. *Journal of Parallel and Distributed Computing*, 68(5):686–702, 2008.

[82] Anurag Singla and Christopher Rohrs. Ultrapeers: Another step towards Gnutella scalability. Gnutella Developers Forum, 2002.

[83] Yee Jiun Song, Venugopalan Ramasubramanian, and Emin Gun Sirer. Optimal resource utilization in content distribution networks. Technical report TR2005-2004, Cornell University, 2005.

[84] Kunwadee Sripanidkulchai. The popularity of Gnutella queries and its implication on scaling. `http://www.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html`, 2001.

[85] Kunwadee Sripanidkulchai, Bruce M. Maggs, and Hui Zhang. Efficient content location using interest-based locality in P2P systems. In *Proceedings of the 22nd IEEE International Conference on Computer Communications (INFOCOM)*, pages 2166–2176, 2003.

[86] Tyron Stading, Petros Maniatis, and Mary Baker. P2P caching schemes to address flash crowds. In *Proceedings of the 1st International Workshop on P2P Systems (IPTPS)*, volume 2429 of *LNCS*, pages 203–213. Springer, 2002.

[87] Konstantinos Stamos, George Pallis, and Athena Vakali. Integrating caching techniques on a content distribution network. In *Proceedings of the 10th East European Conference Advances in Databases and Information Systems (ADBIS)*, volume 4152 of *LNCS*, pages 200–215. Springer, 2006.

[88] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust P2P system to handle flash crowds. In *Proceedings of the 10th IEEE International Conference on Network Protocols (ICNP)*, page 226, 2002.

[89] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: a scalable P2P lookup service for Internet applications. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.

[90] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC)*, pages 189–202, 2006.

[91] Akamai Technologies. Fast Internet content delivery with FreeFlow. White paper, 1999. `http://www.cs.purdue.edu/homes/cs536/lecture/freeflow.pdf`.

[92] Akamai Technologies. Akamai - the business Internet - a predictable platform for profitable e-business. White paper, 2004. `http://www.akamai.com/dl/Whitepapers/Akamai_Business_Internet_Whitepaper.pdf`.

[93] Spyros Voulgaris, Daniela Gavidia, and Maarten Steen. Cyclon: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[94] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. In *Proceedings of the 11th International Conference on Parallel and Distributed Computing (Euro-Par)*, volume 3648 of *LNCS*, pages 1143–1152. Springer, 2005.

[95] Chen Wang, Li Xiao, Yunhao Liu, and Pei Zheng. DiCAS: An efficient distributed caching mechanism for P2P systems. *IEEE Transactions Parallel Distribed Systems*, 17(10):1097–1109, 2006.

[96] Jia Wang. A survey of Web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.

[97] Xiaoyu Wang, Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Buddyweb: A P2P-based collaborative web caching system. In *Revised papers from the NETWORKING Workshops on Web Engineering and Peer-to-Peer Computing*, volume 2376 of *LNCS*, pages 247–251. Springer, 2002.

[98] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 5–14, 2002.

[99] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.

[100] Shelley Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 11–20, 2001.

Contents

1 Introduction **3**

2 Insights on Content Distribution Netwoks **4**
2.1 Background on Web Caching 4
2.2 Overview of CDN 5
2.2.1 Replication and Caching in CDN 7
2.2.2 Location and Routing in CDN 8
2.3 Requirements and Open Issues of CDN 9

3 P2P Systems **10**
3.1 Overview of P2P Systems 11
3.2 Unstructured Overlays 12
3.2.1 Decentralization Degrees 12
3.2.2 Decentralized Routing Techniques 14
3.2.3 Behavior under Churn and Failures 16
3.2.4 Strengths and Weaknesses 17
3.3 Structured Overlays 17

3.3.1 DHT Routing 17
3.3.2 Behavior under Churn and Failures 20
3.3.3 Strengths and Weaknesses 22
3.4 Requirements of P2P Systems 22

4 Recent Trends for P2P Content Distribution 23
4.0.1 Trend 1: Locality-Based Overlay Matching 24
4.0.2 Trend 2: Interest-Based Topology Matching 26
4.0.3 Trend 3: Gossip Protocols as Tools 27
4.0.4 Trend 4: P2P Overlay Combination 31
4.0.5 Challenges to Be Met 33
4.0.6 Discussion 33

5 P2P Content Distribution Systems 34
5.1 Overview 35
5.2 P2P File Sharing 36
5.2.1 Inherent Properties 37
5.2.2 Indexing Approaches 38
5.2.3 Discussion 43
5.3 P2P CDN 44
5.3.1 Insights into Caching and Replication 45
5.3.2 Deployed Systems 46
5.3.3 Centralized Approaches 47
5.3.4 Unstructured Approaches 48
5.3.5 Structured Approaches 48
5.3.6 Discussion 52

6 Conclusion 52

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399