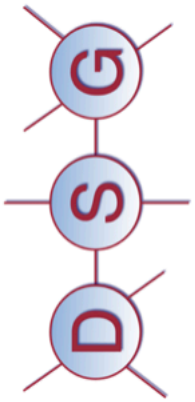
Peer-to-Peer Systems

v.1.0

Michele Amoretti
Distributed Systems Group
Department of Engineering and Architecture
University of Parma
Italy

michele.amoretti@unipr.it
http://dsg.ce.unipr.it/people/michele-amoretti/



Contents

1 Introduction to Peer-to-Peer Systems **2**

2 State Variables **3**
- 2.1 Resource Distribution . . . 4
- 2.2 Resource Popularity . . . 5

3 Dynamics of Peer-to-Peer Networks **6**
- 3.1 Pareto lifetimes . . . 6
- 3.2 Exponential lifetimes . . . 7

4 Design Issues **8**
- 4.1 Effectiveness and Efficiency . . . 8
- 4.2 Security . . . 9

5 Design Strategies for Overlay Schemes **13**
- 5.1 Hybrid Overlay Schemes . . . 14
- 5.2 Decentralized Overlay Schemes . . . 15
- 5.3 Layered Overlay Schemes . . . 19
- 5.4 Standardization Processes . . . 19

6 Popular Overlay Schemes **23**
- 6.1 Content Sharing . . . 23
- 6.2 Distributed Storage . . . 37
- 6.3 Parallel and Distributed Computing . . . 38
- 6.4 Blockchain . . . 38
- 6.5 VoIP and Multimedia Streaming . . . 39
- 6.6 Gaming . . . 40
- 6.7 Education and Academia . . . 40
- 6.8 Internet of Things . . . 41

7 Adaptivity in Peer-to-Peer Systems **42**

Distributed Systems Group
Department of Engineering and Architecture
University of Parma
Parco Area delle Scienze 181a
43124 Parma
Italy
http://dsg.ce.unipr.it

Abstract

In distributed computing, the peer-to-peer (P2P) paradigm enables two or more entities to collaborate spontaneously in an overlay network of equals (peers) by using appropriate information and communication schemes without the need for central coordination. The key concept of the P2P paradigm is leveraging idle resources to do something useful, based on a collaborative approach.

Nowadays, P2P is a well understood alternative to the client/server paradigm. File sharing and multimedia streaming are still the P2P killer applications. However, P2P is also widely adopted in the Internet of Things domain. Last but not least, it is a pillar of the blockchain technology.

The purpose of this essay is to provide the reader with effective design tools for building robust and scalable P2P systems.

1 Introduction to Peer-to-Peer Systems

A peer-to-peer (P2P) network is a complex system whose elements (peer nodes, or simply nodes or peers) are collectors, providers and consumers of resources (CPU cycles, storage, bandwidth, etc.). Resource sharing is based on the collaboration among peers, each one having a limited view of the system and contributing to the implementation of distributed control algorithms. The environment in which P2P networks operate is the set of users, that directly or indirectly act on peers and on resources that each node can use (running environment) and share with other nodes.

The activities of a peer-to-peer system are driven by the environment and by internal feedbacks (Fig. 1). The local environment is perceived by the peer by means of its input interfaces and sensors, providing direct messages from users, but also contextual information. The peer and its neighbors are part of the P2P system which is immersed in the environment. The response of the peer to the inputs that come from the local environment is usually contingent on interactions with neighbors (which in turn may involve their neighbors, etc.). The other way round, a peer can receive requests from its neighbors, and its response in general may depend and affect its local environment.

Thus, environmental inputs usually target a very limited number of peers. When a peer receives an input (from the environment or from other peers), its internal structure maps the input to an output. The mapping process could require the peer to cooperate with other peers, exchanging messages in order to discover and eventually consume resources. For example, if a peer receives the request for a file, it firstly searches its resources and eventually propagates the requests to one or all its neighbors (depending on its internal search algorithm). In any case, *localized reactions follow localized inputs*. Examples of environmental inputs are:

- connection of the peer to the network
- disconnection of the peer from the network
- search queries for resources
- job submissions
- events detected by sensors

We propose to distinguish between three kinds of peer-to-peer systems:

1. Peer-to-peer systems in which each peer is owned by a user, which interacts with it by means of a user interface. This is the case of file sharing platforms, but also collaborative applications such as Skype [67]. These peer-to-peer systems are usually highly dynamic, since peers come and go with users.
2. Peer-to-peer networks in which autonomous peers manage sensors and actuators to provide services to users. These systems are usually contextualized within the fields of ubiquitous computing and ambient intelligence (AmI).
3. Hybrid peer-to-peer systems in which some peers are owned and managed by users, while other peers are autonomous.

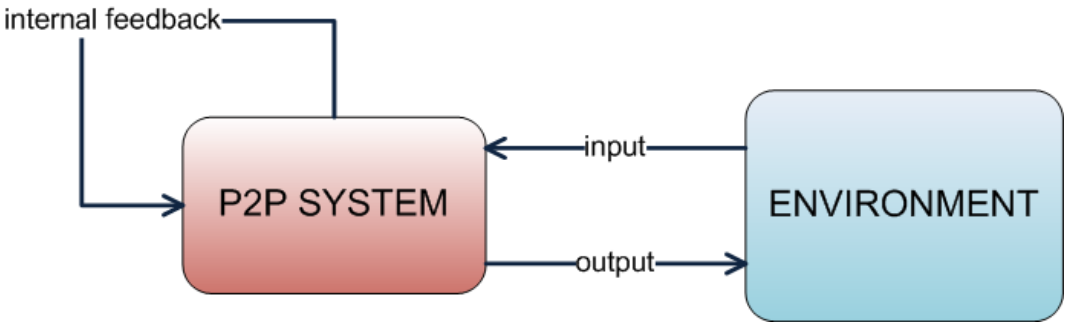


Figure 1: A P2P system and its environment.

2 State Variables

Modeling peer-to-peer networks is challenging. Many variables need to be characterized as *stochastic processes*, which means that even if the initial condition (or starting point) is known, there are many possibilities their values might go to, but some paths are more probable than others.

In general, one fundamental step for the construction of a dynamic model for a system is the definition of the **state variables**, which are the smallest possible subset of system variables that can represent the entire state of the system at any given time. State variables must be linearly independent; a state variable cannot be a linear combination of other state variables.

In this section we formalize the basic variables which characterize a P2P system.

A P2P system is a network of peers. When joining the network, each peer connects (*i.e.* is aware of, not necessarily open a network connection) to other peers according to a predetermined strategy. Connections may change over time, depending on node departures and on other factors.

A peer shares *resources*: bandwidth, cache, CPU, storage, data, services, applications. Sharing implies that resources are discoverable. Peers that share a minimal amount of resources but consume several resources are called *free riders*.

We consider two types of resources:

- *replicable resources* can be moved or copied from one peer to another; examples are data, files;
- *consumable resources* cannot be acquired (by replication), but may only be directly used upon contracting with their hosts; examples are very *basic* resources like storage space, bandwidth, CPU cycles, but also more complex ones, such as applications and services.

Services can be grouped in two categories: *distributed services*, whose execution involves many peers, and *local services*, which are functional units exposed by peers to allow remote access to their local resources.

In user-driven systems, user interests may become characterizing elements for peers. In general, peers search the network for resources, and consume them. Examples of P2P-based applications are file sharing, cycle sharing (with load balancing), multimedia streaming. For details, refer to Section 6.

Peers of the same peergroup use the same message routing algorithm, which can be static (*i.e.* characterized by parameters with fixed values), or adaptive. Messages are exchanged between peers, for example, to publish or retrieve information about resources.

Each peer has a lifetime L that can be modeled considering the following factors: the role of the peer in the system, the availability of resources, and unpredictable events (*e.g.* hardware failures). A study on dynamics of P2P network is proposed in the following Section 3.

Variables that can be *measured* by the peer itself:

- usage of basic shared resources (cache, bandwidth)
- response time of neighbors
- normalized query hit ratio (QHR)
- etc.

Variables that can be *set* by the peer itself:

- size of the routing table (RT)
- RT filling strategy
- message routing algorithm
- upload bandwidth
- download bandwidth

2.1 Resource Distribution

Each peer has a cache filled with a certain number of advertisements describing known local and remote shared resources. We assume that the number of shared non-basic resources owned by the i-th peer ($i = 1, .., N$) is r_i. We also assume that the j-th resource ($j = 1, .., m$) is replicated on n_j nodes. If the total number of resources in the network is r_{tot}, then

$$\sum_{j=1}^{m} n_j = \sum_{i=1}^{N} r_i = r_{tot}$$

Examples of resource distributions are:

- Uniform: $n_j = r_{tot}/m$
- Zipf-like: $n_j \propto 1/j^{\tau}$ with τ close to 1

2.2 Resource Popularity

Users have interests which may be explicitly declared in order to characterize their peer nodes. Radically different approaches emerge from the state of the art, but the common denominator is the tendency to enrich, by means of semantics, the description of easily annotable resources, such as documents or multimedia contents.

The query distribution defines the frequency of queries for individual resources. We assume that there are m resources of interest, and q_j is the (normalized) number of queries issued for the j-th resource (q_j is the relative popularity of resource j):

$$\sum_{j=1}^{m} q_j = 1$$

Examples of query distributions are:

- Uniform: $q_j = 1/m$
- Zipf-like: $q_j \propto 1/j^\tau$ with τ close to 1

In some peer-to-peer systems the distribution of resources depends on the distribution of queries. For example, in file sharing systems, when files are downloaded they may be immediately made available to other peers, thus increasing replication. Examples of resource distributions depending on query distributions are:

- Proportional: $n_j \propto q_j$
- Square-root: $n_j \propto \sqrt{q_j}$

The square-root replication minimizes the overall search traffic [46], and can be obtained with the path replication strategy: when a search succeeds, the resource is stored at all nodes along the path from the requester node to the provider node. This is quite easy if the resource is a file, as in Freenet [20], but difficult for other types of resources.

3 Dynamics of Peer-to-Peer Networks

In the context of peer-to-peer networks, B&D processes can be used to model node arrivals and departures. Nodes in a P2P network do not constitute a population in the traditional sense, because their growth in number is not the result of reproduction of existing nodes. Nodes join or leave a network for many different unpredictable reasons, even if the presence of *interesting resources* is the fundamental motivation for users. Some nodes are always connected, other nodes join to consume resources and then leave, etc. We consider B&D processes characterized by having the arrival rates and the service rates independent of the state. However, the following analysis applies to any arrival/departure process, as long as the number of nodes N stays sufficiently large [38].

The lifetime of a peer node is a time interval that we call L. Residual lifetime R is the remaining lifetime of a neighbor when a node joins the network. Assuming that the number of nodes N is large and the system has reached stationarity, the CDF of residual lifetimes is given by:

$$F_R(x) = P(R \leq x) = \frac{1}{E[L]} \int_0^x (1 - F_L(z))dz \tag{1}$$

If there are N nodes at time t, then the *doubling time from time* t is the time that elapses before N additional nodes arrive. The *halving time from time* t is the time required for half of the nodes alive at time t to depart. The *half-life from time* t is the smaller of the doubling and halving times from time t. Finally, the half-life of the entire system is the minimum half-life over all time t.

In the following we consider two different lifetime models, one that applies to human-based P2P systems, the other that may be applied to systems of non-human devices and software agents.

3.1 Pareto lifetimes

It has been observed that the distribution of user lifetimes in real P2P systems is often heavy-tailed (*i.e.*, Pareto) [9, 66], where most users spend minutes per day browsing the network while a handful of other peers exhibit server-like behavior and keep their computers logged in for weeks at a time.

A model which allows arbitrary small lifetimes is the shifted Pareto distribution, whose CDF is

$$F_L(x) = 1 - (1 + \frac{x}{\beta})^{-\alpha}, x > 0, \alpha > 1, \beta > 0 \tag{2}$$

where the scale parameter β can change the mean of the distribution without affecting its range $(0, \infty]$. The mean of this distribution is $E[L] = \beta/(\alpha - 1)$.

In this case, residual lifetimes have a distribution which is more heavy-tailed than $F_L(x)$ [82]:

$$F_R(x) = 1 - (1 + \frac{x}{\beta})^{1-\alpha} \tag{3}$$

This outcome is not surprising, because heavy-tailed distributions exhibit memory, as

$$P(L > x + t | L > t) = 1 - F_L(x + t | L > t) = \frac{1 - F_L(x + t)}{1 - F_L(t)} = (1 + \frac{x}{\beta + t})^{-\alpha}$$

which is different from

$$P(L > x) = 1 - F_L(x) = (1 + \frac{x}{\beta})^{-\alpha}$$

In this context, having memory means that nodes who survived for some time $t > 0$ are likely to remain online for longer time periods, with respect to arriving nodes. In fact, the more a supernode is aged, the longer it is expected to survive.

3.2 Exponential lifetimes

Peers' lifetimes are described by an exponential random variable, whose CDF is

$$F_L(x) = 1 - e^{-\lambda x} \tag{4}$$

with expected lifetime $E[L] = 1/\lambda$ and variance $1/\lambda^2$.

Residual lifetimes have the same CDF of L:

$$F_R(x) = F_L(x)$$

This is a consequence of the memoryless property of the exponential distribution:

$$P(L > x + t|L > t) = 1 - F_L(x + t|L > t) = \frac{1 - F_L(x + t)}{1 - F_L(t)} = \frac{e^{-\lambda(t+x)}}{e^{-\lambda t}} = e^{-\lambda x}$$

which is equal to

$$P(L > x) = 1 - F_L(x) = e^{-\lambda x}$$

The doubling time is $1/\lambda$, while the halving time and half-life are both $ln2/\lambda$ [42].

4 Design Issues

The operation of any peer-to-peer system relies on a network of peer software/hardware nodes, and connections (links) between them. This network is formed on top of - and independently from - the underlying physical computer (typically IP) network, and is thus referred to as an overlay network. The topology, structure, and degree of centralization of the overlay network, and the message routing and location mechanisms it employs for messages and resources are crucial to the operation of the system, as they affect its scalability, security, fault tolerance, and self-maintainability.

In recent years, the research on P2P systems has focused on designing robust overlay schemes (defining the rules for bootstrapping, connectivity, message routing, caching), usually targeting specific applications. In the following we briefly recall the most intriguing P2P design challenges, in order to have a base for the categorization of overlay schemes proposed in Section 5 and for the survey of the most popular ones presented in Section 6. In our opinion, a P2P system should be effective, efficient and secure, both in general and with respect to its specific application(s).

4.1 Effectiveness and Efficiency

The focus of effectiveness is the achievement of a goal, by performing the right actions. Efficiency means doing things in the most economical way (good input to output ratio). What is effective is not necessarily efficient, in P2P systems. Here we illustrate the main issues that affect the effectiveness and efficiency of P2P architectures.

The **scalability** of an overlay scheme measures its effectiveness and efficiency, with respect to the target application(s), when applied to large situations (*e.g.* large workloads or large number of participating nodes). A distributed system should be inherently more scalable if using the P2P paradigm, rather than the client/server approach. But some P2P overlay schemes scale better than others, with respect to resource discovery effectiveness and performance, bandwidth occupancy, etc. For example, a message routing protocol is considered scalable with respect to network size, if the number of message propagations that are necessary to find a resource grows as $O(logN)$, where N is the number of nodes in the network.

The effectiveness and efficiency of P2P protocols for resource sharing usually depends on how peer are connected. One key operation is **bootstrapping**, the initial discovery of other nodes participating in the network. Nascent peers need to perform such an operation in order to join the network. Bootstrapping usually includes operations needed to repair overlays that have split into disconnected subgraphs [21]. Another important operation is **connectivity management**, *i.e.* the maintenance of connections or exchange of topology information for peers that are already connected to the network at large.

Search performance and **consistency** are two important measures for the sharing of dynamic contents (*e.g.* in P2P storage systems). Search performance concerns how fast the users locate and obtain copies of requested resources (time complexity) and how many nodes must be involved in that process (space complexity). Consistency concerns how old the acquired data (resource descriptions or

shared content) are with respect to the actual available resources.

Stability refers to the ability of the overlay scheme to guarantee the correctness of its functionalities regardless of the churn rate of the network. The churn rate is a measure of the number of peers joining and leaving the system over a specific period of time. Users of the peer-to-peer system join and leave the network randomly, which makes the overlay network dynamic and unstable in nature. This dynamical behavior of the peers frequently partitions the network into smaller fragments which results in the breakdown of communication among peers. Thus, design and tuning of the overlay scheme should be made taking into account different churn rate profiles, depending on the distributed application the scheme should support.

Finally, **load balancing** is the measure of how fair is the distribution of computational, storage and bandwidth consumption among the nodes participating in the network. Whilst this issue is well defined and basically solved for systems with relatively simple search paradigms, only few known solutions are appropriate or applicable for similarity search networks (*e.g.* supporting semantic or fuzzy resource discovery).

Another issue that potentially can hinder the deployment of large-scale P2P applications is **asymmetric bandwidth** in the access network. In particular, the uploading capability of each peer can become a bottleneck in the system. This can significantly impact ISP (Internet service providers) and how ISP perform traffic dimensioning. Moreover, a large portion of the Internet bandwidth is occupied by P2P applications, where many ISP have enforced traffic engineering mechanisms, in particular for inter-domain traffic. For content sharing, this implies considerable slowdown in performance; but for streaming applications, this can be fatal. Finally, NAT and firewalls can impose fundamental limitations on the pair-wise host connectivity in the overlay network. It is well-known that a significant portion of broadband users experience NAT or firewall problems, and this requires particular attention.

4.2 Security

Peer-to-peer architectures present a particular challenge for providing high levels of privacy, confidentiality, integrity, and authenticity, due to their open and autonomous nature. Preserving integrity and authenticity of resources means safeguarding the accuracy and completeness of data and processing methods. Unauthorized entities cannot change data; adversaries cannot substitute a forged document for a requested one. Privacy and confidentiality mean ensuring that data is accessible only to those authorized to have access, and that there is control over what data is collected, how it is used, and how it is maintained. A malicious node might give erroneous responses to requests, both at the application level, returning false data, or at the network level, returning false routes and partitioning the network. Moreover, the P2P system must be robust against a conspiracy of a malicious collective, *i.e.* a group of nodes acting in concert to attack reliable ones. Attackers may have a number of goals, including traffic analysis against systems that try to provide anonymous communication, and censorship against systems that try to provide high availability.

Security attacks in P2P systems can be classified into two broad categories: *passive* and *active*. Passive attacks are those in which the attacker just monitors activity and maintains an inert state. The most significant passive attacks are:

- **Eavesdropping**. It involves capturing and storing all traffic between some set of peers searching for some sensitive information (such as personal data or passwords).
- **Traffic analysis**. Here the attacker not only captures data but tries to obtain more information by analyzing its behavior and looking for patterns, even when its content remains unknown.

In active attacks, communications are disrupted by the deletion, modification or insertion of data. The most common attacks of this kind are:

- **Spoofing**. One peer impersonates another, or some outside attacker transforms communications data in order to simulate such an outcome.
- **Man-in-the-middle**. The attacker intercepts communications between two parties, relaying messages in such a manner that both of them still believe they are directly communicating. This category includes data alteration between endpoints.
- **Playback or Replay**. Some data exchange between two legitimate peers is intercepted by the attacker in order to reuse the exact data at a later time and make it look like a real exchange. Even if message content is encrypted, such attacks can succeed so long as duplicate communications are allowed and the attacker can deduce the effect of such a repeat.
- **Local Data Alteration**. This goes beyond the assumption that attacks may only come from the network and supposes that the attacker has local access to the peer, where he/she can try to modify the local data in order to subvert it in some malicious way.
- **No-forwarding**. An important issue that looms over P2P networks is blocking and throttling of P2P traffic. According to a 2007 Internet study [28], 69% of Internet traffic in Germany is P2P, with HTTP way behind at 10%. Given the staggering proportion of Internet traffic accounted for by P2P applications, especially BitTorrent (from the numbers above, BitTorrent alone accounts for nearly 50% of the Germany's Internet traffic), it is not surprising that ISPs are starting to block ports on which well-known file sharing applications run. For example, in 2007 Comcast started to throttle and drop packets of BitTorrent traffic, effectively blocking its customers from running the software [68].
- **Free Riding**. So far we generally assumed that peers are willing to collaborate with one another. However, this is not always true in reality. In a typical P2P system, some users tend to be selfish, for example, saving resources and taking free rides (such users are called free riders). Potentially this has a serious impact on the performance of cooperative peers. Some researchers

have explored the benefits of enabling virtual communities to self-organize and introduce incentives as a resource sharing and cooperation, arguing that what is missing from today's peer-to-peer systems, should be seen both as a goal and a means for self-organized virtual communities to be built and fostered [3]. Ongoing research efforts for designing effective incentive mechanisms in P2P systems, based on principles from game theory are beginning to take on a more psychological and information processing direction. However, it is not possible to create a general-purpose framework. For example, in file sharing, credit-based or reputation-based mechanisms can be adopted, but this is difficult to do for streaming applications. The reason is that in live streaming applications it is difficult to collect credits or a reputation due to the timing requirement. How to design an incentive mechanism to combat the possible misbehaving peers remains an open problem. It has been shown that the tit-for-tat strategy (adopted by some file sharing applications) does not work for live streaming applications [40]. This is because each peer only maintains streaming data in a window with a short time period, which implies that it is nearly impossible for a descendant of a peer in one sub-stream to be an ancestor of other sub-stream.

- **Distributed Denial of Service (DDoS)**. In a traditional denial-of-service (DoS) attack, a server is usually the target of massive connections, rendering the server inoperable. In a P2P network, attackers can make use of the querying nature of P2P networks to overload the network. In the case of the query flooding P2P network, the attack is straightforward: simply send a massive number of queries to peers, and the resulting broadcast storm will render portions of the network inoperable. More recently, attacks can harness the P2P network as an agent to attack some other target, such as web sites. Essentially, peers in the network are subverted to request files from a target, overwhelming the victim with enormous bandwidth usage. An example of this kind of attack surfaced in 2007 in the Direct Connect network with users using the DC++ file sharing application [56].

- **Network Poisoning**. Another approach towards attacking a P2P network is to inject useless data (poison) into the system. Since P2P networks must implement a lookup service in some way, an attacker can inject large amounts of useless lookup key-value pairs into the index. Bogus items in the index could slow down query times or, worse, yield invalid queries results. In fact, poisoning a P2P network has already been witnessed on the Internet as large publishing organizations attempt to lessen the potential losses of pirated media by attacking the FastTrack P2P network [41]. Poisoning can also be used as fodder for DDoS attacks. This can be accomplished in two ways, by index poisoning or route table poisoning. In index poisoning, fake records are inserted into the index pointing to a target IP and port number. When a peer goes to search for a resource, it would receive bogus location information from a poisoned index, either from a central directory or from another peer. The requesting peer then makes a connection to the target, perhaps confusion the target or, if the target accepts the connection, a TCP-connection DDoS comes

into effect. In route table poisoning, the attack leverages the fact that almost all P2P clients need to maintain some kind of routing state of the current peers with which it is connected. The attacker dupes peers into adding bogus neighbors into each peer's route table, and in some cases, this as simple as making an announce message pointing to the target. The result is that the target receives a flood of connection requests, and the target will likely reject them.

Trust management is another challenging problem, defining how to enable a peer to decide whether to trust another peer, in the absence of a central trust managing authority. Trust is important when sharing data or processing power, and crucial for e-commerce applications such as auctioning. According to Gambetta's definition [22], trust (or, symmetrically, distrust) is a particular level of the subjective probability with which a peer will perform a particular action, both before we can monitor such action (or independently of its capacity of ever to be able to monitor it) and in a context in which it affects our own action. It is important to remark that in general trust is non-transitive [1], *i.e.* if peer A trusts peer B and peer B trusts peer C, it does not follow that peer A trusts peer C. Moreover, as trust is subjective, peer B and peer C could have different degrees of trust in peer A. Several trust decision strategies have been proposed by the research community.

5 Design Strategies for Overlay Schemes

Current P2P systems do commonly only address a subset of the issues that are listed in section 4, for the large number of trade-offs and constraints due to the different dimensions and purposes such systems are demanded to address. In general, each P2P system is based on an overlay scheme, which defines how peers are connected, how messages are propagated among nodes to share resources and information about them, and which security mechanisms are adopted.

The placement of information about shared resources plays an important role in the characterization of an overlay scheme. Information about shared resources can be

- published to a central server
- published to other peers
- locally stored by resource owners and not published

The first approach leads to *hybrid overlay schemes* (based on the **Hybrid Model - HM**), so called because they are based on the client/server paradigm in resource publication and discovery, while the peer-to-peer approach is used for resource consumption. The other approaches lead to *decentralized overlay schemes*, only relying on local information available at each node (such networks are often referred as "pure" P2P systems). Decentralized P2P systems can be divided in two groups, depending on the topology awareness of peers. A decentralized P2P overlay is unstructured (based on the **Decentralized Unstructured Model - DUM**) if links among peers (being them actual or potential connections) can be represented by a random graph, whose characteristics are unknown to the peers, and not relevant to their message routing strategies. On the contrary, a decentralized P2P overlay is *structured* (based on the **Decentralized Structured Model - DSM**) if its topology is controlled and shaped in a way that resources (or resource advertisements) are placed at appropriate locations.

To improve the performance (with respect to scalability, lookup performance and stability) of P2P networks, layered overlay schemes (based on the **Layered Model - LM**) have been studied and implemented [18,19]. Such overlays are characterized by interacting layers, each one being organized according to one of the "flat" models (HM, DUM, or DSM).

The table in figure 2 summarizes the advantages and disadvantages of HM, DUM and DSM. The success that the Hybrid Model has obtained in recent years (Napster, BitTorrent, etc.) is clearly justified: almost all effectiveness and efficiency attributes have good values, with the exception of scalability, and many security issues are solved.

A detailed discussion is provided in the following sections, where for each overlay scheme we define its main general characteristics, considering the strengths and flaws, then we present recent related patents.

	HM	DUM	DSM
Scalability	Low	Medium	High
Bootstrapping	Simple	Complex	Simple
Connectivity Mgmt	Simple	Complex	Complex
Time Complexity	*1*	*O(N)*	*O(logN)*
Space Complexity	*1*	*O(d)*	*O(logN)*
Consistency	High	Low	High
Stability	High	High	Low
Load Balancing	No	No	Yes
Eavesdropping	No	Yes	Yes
Traffic Analysis	Yes	Yes	No
Spoofing	No	Yes	Yes
Man-in-the-middle	Yes	Yes	Yes
Playback or Replay	Yes	Yes	Yes
Local Data Alteration	Yes	Yes	Yes
No-forwarding	No	Yes	No
Free Riding	Yes	Yes	Yes
DDoS	No	Yes	No
Network Poisoning	No	Yes	No

Figure 2: Contrast between Hierarchical Model (HM), Decentralized Unstructured Model (DUM) and Decentralized Structure Model (DSM). The number of nodes is N, and each node has d neighbors (it is the average value, in case of DUM). Attributes are divided in two groups: effectiveness/efficiency (green), and security (orange).

5.1 Hybrid Overlay Schemes

In peer-to-peer systems based on the HM model, illustrated in figure 3, peers connect to one or more central servers in which they publish information about the resources they offer for sharing. Each central server maintains, for each resource, the list of owners, in some case partially replicating the lists of other known servers. Upon request from a peer, the central directory provides the list of peers that match the request, or the best peer in terms of bandwidth and availability. Further interactions occur directly between the resource provider and the consumer. If the central server is not unique, queries may also be forwarded to neighbor servers.

The success of Napster, eMule and BitTorrent file sharing systems (discussed in Section 6) demonstrates the effectiveness of the Hybrid Model. The main reason is that HM-based peer-to-peer systems are only moderately affected by security issues. Centralized servers manage file transfer and allow more control which makes it much harder faking IP addresses, port numbers, etc. Unfortunately, for the same reason anonymity is difficult to achieve.

On the other hand, HM-based overlay schemes are absolutely not suitable for P2P streaming or gaming applications, since central servers would quickly become bottlenecks. Indeed, most servers dramatically lower their efficiency, measured in

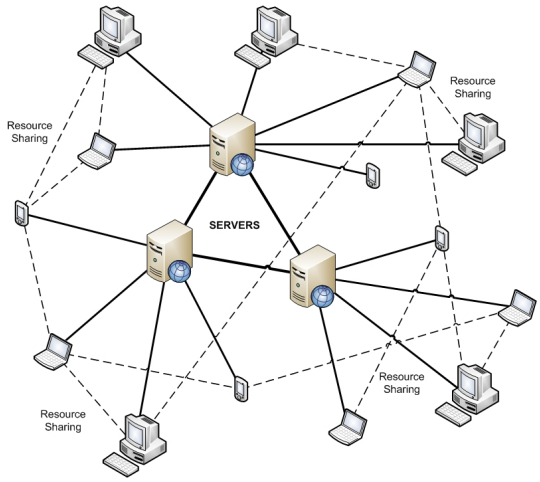


Figure 3: Typical peer-to-peer system based on the Hybrid Model (HM). Dashed lines represent the peer-to-server and server-to-server exchange of information about shared resources. Continuous lines represent peer-to-peer resource sharing (e.g. content upload/download).

terms of responsiveness, when the number of connected peers drastically increases (to several hundred thousands).

5.2 Decentralized Overlay Schemes

As illustrated in figure 4, in the Decentralized Unstructured Model (DUM) information about resources is distributed among peers. For applications like distributed gaming and multimedia streaming, the (DUM) is the most effective, provided that control messaging does not waste network bandwidth. Each peer propagates requests to directly connected peers, according to some strategy, *e.g.* by flooding the network with messages, a strategy that consumes a lot of network bandwidth, and hence does not prove to be very scalable, but it is efficient in limited communities such as company networks. To improve scalability, caching of recent research requests can be used, together with probabilistic flooding (*i.e.* each peer propagates messages to a random number of neighbors). Moreover, if a unique identity is added to the message, nodes can delete reoccurrences of the same message and stop loops from forming. In large networks it may be necessary to include some kind of time-to-live (TTL) counter to stop the message flooding the network.

Peer-to-peer architectures based on the Decentralized Structured Model (DSM), illustrated in figure 5, are characterized by a globally consistent protocol ensuring that any node can efficiently route a search to some peer that has the desired resource, even if the resource is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured P2P network is the distributed hash table (DHT). Each resource must be identified by a unique key and associated to a description (including a pointer to the resource owner): key and description form a $< key, value >$ pair associated to the resource. Each peer is assigned a random ID in same space of resource keys, and it is respon-

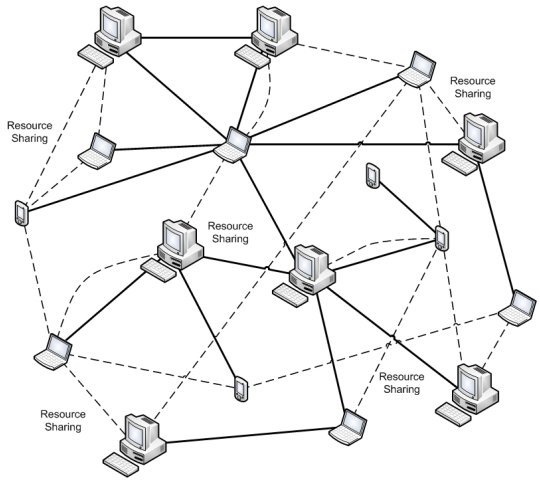


Figure 4: Typical peer-to-peer system based on the Decentralized Unstructured Model (DUM). Dashed lines represent the peer-to-peer exchange of information about shared resources. Continuous lines represent peer-to-peer resource sharing (e.g. content upload/download).

sible for storing $< key, value >$ pairs for a limited subset of the entire key space. When a resource description is published, the peer routes the associated key/value pair towards the peer with the ID that is most similar to the block ID. This process is repeated until the nearest peer ID is the current peer's ID. In DSM-based overlay schemes, the responsibility of storing information about shared resources is much more fairly distributed among peers than in DUM-based ones. When a peer makes a request for a resource, the query is propagated towards the peer with the ID which is most similar to the resource ID.

Beyond basic routing correctness, two important constraints on the topology are to guarantee that the maximum number of hops in any route (route length) is low, so that requests complete quickly; and that the maximum number of neighbors of any node (maximum node degree) is low, so that maintenance overhead is not excessive. Of course, having shorter routes requires higher maximum degree. Some common choices for maximum degree and route length are as follows, where n is the number of nodes in the DHT, using Big O notation:

- degree $O(1)$, route length $O(N)$
- degree $O(\log N)$, route length $O(\log N / \log \log N)$
- degree $O(\log N)$, route length $O(\log N)$
- degree $O(\sqrt{N})$, route length $O(1)$

The third choice is the most common, even though it is not quite optimal in terms of degree/route length tradeoff, because such topologies typically allow more flexibility in choice of neighbors. Many DHTs use that flexibility to pick neighbors which are close in terms of latency in the physical underlying network.

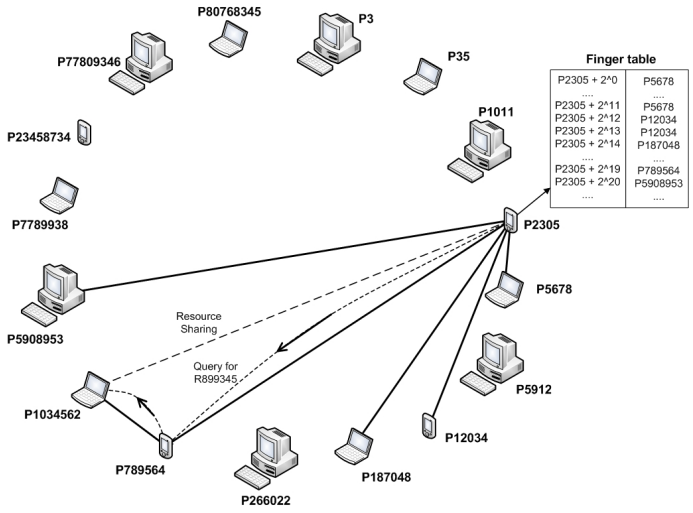


Figure 5: Typical peer-to-peer system based on the Decentralized Structured Model (DSM)-this one in particular recalling Chord's ring. Dashed lines represent the peer-to-peer propagation of queries for shared resources. Continuous lines represent peer-to-peer resource sharing (e.g. content upload/download). Each peer owns a routing table which targets a very limited number of other peers (chosen according to a deterministic strategy). The routing table serves also for publishing information about shared resources (if the advertising strategy is fair, such information is evenly distributed among peers - on the average, all peers have the same amount of responsibility).

Decentralization (either unstructured or structured) introduces new problems in terms of security and trust into these systems. Most existing approaches perform well in cooperative scenarios, but are doomed if this assumption does not hold. Also, the large number of participants in these systems raises new questions, such as whom to trust and cooperate with or how to ensure basic security properties such as confidentiality, authenticity or non-repudiation, and how to support these functionalities without introducing centralization again for being able to meet these requirements. Two basic approaches for securing decentralized P2P systems are *message encryption* and *peer anonymity*.

By encrypting P2P traffic, the hope is that not only will the data be safely encrypted, but more importantly, the P2P data stream is encrypted and not easily detectable. With the actual connection stream completely encrypted, it becomes much harder for the P2P traffic to be detected, and, thus, attacked, blocked, or throttled.

With peer anonymity, the identity of nodes and users is protected on the network, something that encryption alone cannot ensure. While true anonymity cannot really exist on a network, an anonymous P2P provides enough anonymity such that it is extremely difficult to find the source or destination of a data stream. It does this by making all peers on the network universal senders and universal receivers, thus making it practically impossible to determine if a peer is receiving a chunk of data or simply passing it through. The majority of anonymous peer-to-peer systems are

"friend-to-friend" networks. These are P2P networks in which each peer (node) only connects to a small number of other nodes. Only the direct neighbors of a node know its IP address. Communication with remote nodes is provided by sending messages hop-to-hop across this overlay network. Routing messages in this way allows these networks to trade efficient routing for anonymity. There is no way to find the IP address of a remote node, and direct neighbors can achieve a level of anonymity by claiming that they are just forwarding requests and files for other nodes. In anonymous DUM-based overlay networks, the TTL counter is usually probabilistic (as determinism would allow attackers to find the sender/receiver), *e.g.* the time-to-live is reduced by a value proportional to the number of results found at a node and the number of connections the search message is forwarded to.

There is a danger that an attacker will be able to spy on the activity of their direct neighbors, and thus find out which files the neighbor is requesting or offering. Some systems contain faults that leak this information while others allow an attacker to be up to 50% certain of what their neighbor is doing. None of the current systems try to make it hard for an attacker to work out whether or not someone is running the P2P software. A list of research papers on anonymity can be found at the Free Haven site [19]. Freenet [20] is probably the most famous anonymity-oriented P2P network (see Section 6 for details).

Against network poisoning, DSM protocols have a mechanism to remove stale peers from the routing table, updating it constantly. Thus, after the burst of traffic to the target, the target is removed from the route tables of connecting peers [52].

To cope with the attacks illustrated in Section 4, other than encryption and peer anonymity, the security policies adopted at the overlay P2P network level usually consist of key management, authentication, admission control, and authorization. In the context of decentralized P2P systems, several trust management approaches have been proposed. The approach we prefer is based on the cooperation of peers in order to define the reputation of resource providers. We devote section 5 to the discussion of such solution, after the presentation of the most popular P2P overlay schemes and applications provided in Section 6.

A general advantage of unstructured overlays is that they can support complex queries more effectively than structured overlays. Due to the nature of hash functions, fuzzy search algorithms required for keyword based searching are hard to implement in DHTs because a small change in the input will result in a completely different output and thus another key and not a nearby one. Thus only precise lookups, *i.e.* based on unique content identifiers can be performed on a DHT without large overhead.

It is also commonly believed that structured graphs are more expensive to maintain than unstructured graphs and that the constraints imposed by the structure make it harder to exploit heterogeneity to improve scalability. For these reasons, many research studies and concrete projects have targeted layered (LM-based) systems, outperforming flat architectures [5, 36].

5.3 Layered Overlay Schemes

In layered architectures, peers are grouped into layers, each one being organized according to a flat overlay scheme (HM, DUM or DSM). The interaction among layers is usually defined by an application-specific protocol. A typical example is the 2-layered architecture, in which peers with higher bandwidth and process capacity act as supernodes, assuming the responsibility of propagating messages, while peers with less capacities (leaf nodes) are only resource providers and consumers, and need to connect to the supernode layer in order to publish/discover shared resources. An example of 2-layered scheme is illustrated in figure 6.

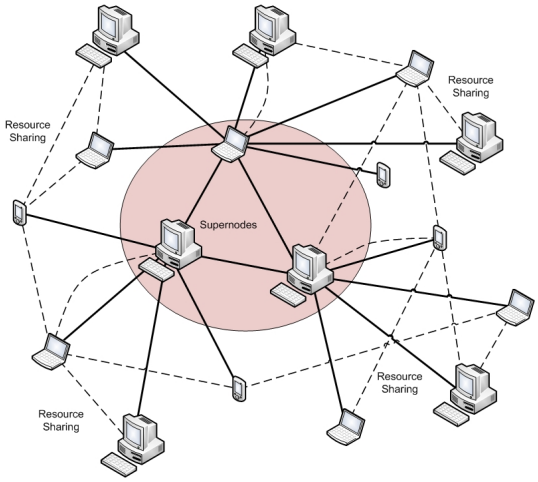


Figure 6: Example peer-to-peer system based on a 2-layered mode. Dashed lines represent the exchange of information about shared resources - in this function, each leaf node is allowed to interact only with its supernode. Continuous lines represent peer-to-peer resource sharing (e.g. content upload/download).

5.4 Standardization Processes

During the last ten years, many different P2P architectural blueprints and protocols have been proposed. Unfortunately each developer promoted his/her brainchild targeted to a specific application and with a restricted set of functionalities, without any sort of significant coordination and standardization effort. This fact has often produced overlapping and redundancy of functionalities and a complete lack of interoperability.

Probably the most significant work in the direction of an unique signaling platform and protocol suite is the JXTA project [76], originally conceived by Sun Microsystems, and designed with the participation of a growing number of experts from academic institutions and industry. The main goal of JXTA is to define a generic P2P network overlay usable to implement a wide variety of applications and services. The JXTA platform provides core building blocks (IDs, advertisements, peergroups, pipes) and a default set of core policies, which can be replaced if necessary. The overlay network is 2-layered, with two primary node types:

- *rendezvous super-peers* (supernodes), which are connected according to a DUM-based scheme; supernodes have agreed to cache pointers (indices) to edge peers that cache the corresponding advertisement (the XML description of a shareable resource), and are able to propagate queries.
- *edge peers* (leaf nodes), which are able to send query/reply messages but not to propagate queries.

Each edge peer is connected to one rendezvous super-peer. Each rendezvous has a Rendezvous Peer View (RPV), which is an ordered list of other known rendezvous.

Shareable entities (resources and services) are described by XML documents named *advertisements*, whose entries are attribute/value pairs. Message routing for sharing and searching advertisements in the JXTA rendezvous network is based on two components: the *Shared Resource Distributed Index (SRDI)*, and the *loosely-consistent DHT walker*. The SRDI module implemented in each JXTA peer is used on one hand to extract entries from advertisements and push them to the network, and on the other hand for lookups. The walker is used for routing when no index information is locally available. The process is illustrated in figure 7, for both the ideal case of complete supernode network, and the more common case of uncomplete supernode network.

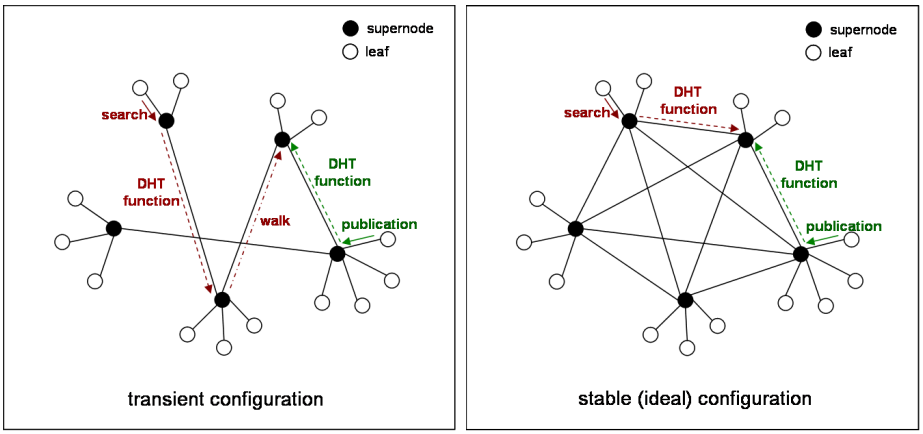


Figure 7: JXTA routing with the SRDI strategy.

When an advertisement is published by an edge peer (E1), its entries (which are attribute-value pairs) are sent to the connected supernode (R1). Supernodes store the entries in dynamic indexes including also, for each entry, the ID of the peer which originated them and an expiration time. Moreover, each index entry is replicated from supernode R1 to another supernode in R1's RPV using a DHT function. In details, the 160 bit SHA1 hash address space is evenly divided amongst the ordered RPV (sorted by pid), so an index entry is routed by hashing its value and mapping its location in the RPV (suppose R11). If the RPV is > 3, each index entry is also replicated to the RPV neighbors of R11 ($+1$ and -1 in the RPV ordered list).

Search is based on the same DHT function, but also on a limited range walker to resolve inconsistency of the DHT within the dynamic rendezvous network. Queries are messages which contain advertisement entries (attribute-value pairs). If a query

is sent by an edge peer (E2), it reaches the connected supernode (R2). Once R2 receives the query, it first attempts to match it locally. If a match is not found, then R2 forwards the query to another supernode in R2's RPV (suppose R22) using the DHT function. The more the rendezvous network is near to completeness, *i.e.* the RPV is consistent across all supernodes, the more the DHT-based routing algorithm is efficient. To compensate for any RPV skew, a *limited range walker* is used. For example, suppose R22 fails to match the query, and its RPV is

R20 R21 *R22* R23

Assume that R20 and R21 have the same RPV of R22, while R23 has the following RPV:

R23 R24 R25

Thus an R22 originated limited range walker query is walked to R21 with a TTL of 2, and to R23 with a TTL of 1, where the TTL is adjusted to 2 on R23 and walked to R25. When there is a query hit, the response is forwarded to the query originator (in this case, edge E2).

The more the supernode network is near to completeness, *i.e.* the peerview is consistent across all supernodes, the more the routing algorithm is efficient. The strategy is summarized in algorithm 1.

Algorithm 1 JXTA message routing.

```
1: if (publication) then
2:     save locally
3: end if
4: if (search) then
5:     match locally
6: end if
7: if (leaf peer) then
8:     send message to supernode
9: end if
10: if (supernode) then
11:     if ((message from leaf peer) || (local message)) then
12:         find target supernode neighbor t using DHT function
13:         send message to target t supernode neighbor
14:         if ((publication) && (peerview > 3)) then
15:             send message to t + 1 and t − 1 supernode neighbors
16:         end if
17:     end if
18:     if (message from supernode) then
19:         if (search) then
20:             walk the peerview
21:         end if
22:     end if
23: end if
```

Other important P2P standardization attempts have been done. The Internet Engineering Task Force (IETF) has formed working groups to address specific issues of P2P networking:

- Application-Layer Traffic Optimization (ALTO)[1]
- P2P Session Initiation Protocol (P2PSIP)[2]

P2PSIP is the most active P2P WG, having provided several proposals and a specific signaling protocol [30]. As stated by the P2PSIP working group itself, the goal is to develop protocols and mechanisms for establishing and managing sessions completely handled by peers. The primary objective of such protocols is to provide a common platform for user and resource location in a complete distributed environment. Although it was started for using for end-to-end real-time session initiation, P2PSIP could apply also to any other interactive or transactional applications.

In particular, the P2PSIP group released the Internet draft of REsource LOcation And Discovery (RELOAD),[3] a peer-to-peer (P2P) signaling protocol designed to support a P2PSIP network, but can be utilized by other applications with similar requirements by defining new usages that specify the kinds of data that needs to be stored for a particular application. RELOAD defines a security model based on a certificate enrollment service that provides unique identities. NAT traversal is a fundamental service of the protocol. RELOAD also allows access from "client" nodes that do not need to route traffic or store data for others.

[1] http://tools.ietf.org/wg/alto/
[2] http://tools.ietf.org/wg/p2psip/
[3] https://tools.ietf.org/html/rfc6940

6 Popular Overlay Schemes

Over the last few years, peer-to-peer content sharing systems have enjoyed explosive popularity. Thus, most research efforts have been spent on designing effective, efficient and secure protocols for such kinds of applications. Nevertheless, we must observe that the basic principles of HM, DUM, DSM and LM can be applied to any other kind of distributed application involving peer-to-peer resource sharing, *i.e.*, cooperative advertising, discovery and consumption of resources. In the following we discuss the most popular overlay schemes, grouped by application domain. Cross-domain schemes are highlighted.

6.1 Content Sharing

In P2P content sharing, users can share their content by contributing their own resources to one another. However, since there is no incentive for contributing contents or resources to others, users may attempt to obtain content without any contribution. In the following we describe the most important content sharing protocols, with particular attention to those that motivate users to contribute their resources.

Soulseek In this context, Soulseek [69] is a file-sharing network and application based on the HM scheme. It is used mostly to exchange music, although users are able to share a variety of files. The central server coordinates searches and hosts chat rooms, but it does not actually plays a part in the transfer of files, which takes place directly between the concerned users. Being one of the first HM protocols, it has many limitations with respect to others. In particular, it does not allow multi-source download of files.

Napster Also based on the HM scheme, the Napster [53] protocol was designed with the same purposes of Soulseek, but it gained much more success because of several free implementations, both open source (gnapster, Knapster, etc.) and closed source (the official Napster client), and also several related utilities. Residing on user machines, files never pass through central Napster servers, which provide the ability to search for particular files and initiate a direct transfer between peers. OpenNap [58] extends the Napster protocol to allow sharing of any media type, and the ability to link servers together.

Napster was a phenomenon: the official client went live in september 1999, and by mid-2001 it had over 25 million users. The impact of using Napster for online MP3 file sharing to distribute music upset what had been a well-established business model. Christensen [11] defines two types of technologies: *sustaining technology* and *disruptive technology*. Sustaining technology is that with a developed market. In well-managed companies which supply this technology, the products advance as rapidly as improvements in science and engineering permit - along the lines desired by the customers of the company. Managers carefully determine the desires of their customers and plan engineering development projects to satisfy those customers as quickly and cost-effectively as possible. These companies develop and infuse their work force with ethics, procedures, and goals consistent with this process in their

respective industries. Disruptive technologies, on the other hand, are usually simpler and cheaper than the sustaining technology, but also offer less capability. They do not fit the sustaining market and, typically, provide lower profit margins. For this reason, they are usually shunned by well-managed companies - which are often later destroyed by them.

Napster is a classic example of disruptive technology. With millions of users trading pirated songs, the music industry rushed to put a stop to it. In 2000, A&M records and several other recording companies sued Napster for contributory and vicarious copyright infringement under the US Digital Millennium Copyright Act (DMCA). The District Court ordered Napster to monitor the activities of its network and to block access to infringing material when notified of that material's location. Napster was unable to do this, and so shut down its service in July 2001. Napster finally declared itself bankrupt in 2002 and sold its assets.

In 2003 Napster 2.0 has been released by a division of Roxio Inc. and its servers now offer online music store services, having extensive content agreements with the five major record labels, as well as hundreds of independents.

eDonkey The eDonkey protocol (also known as eDonkey2000 or eD2k), HM-based, allows to create file sharing networks for the exchange of multimedia content. Client programs connect to the network to share files, and publish to servers that can be set up by anyone. Servers act as communication hubs for the clients and allow users to locate files within the network.

In eDonkey, free riders are penalized by the following mechanism. A client ID is provided to a peer by the contacted server at their connection handshake, and it is valid only through the lifetime of a peer-server TCP connection, although in case the peer has a high ID it will be assigned the same ID by all servers until its IP address changes. Identifiers are divided to low IDs and high IDs. A server typically assigns a low ID to a peer if the latter cannot accept incoming connections (for example because it is NATted). Having a low ID restricts the peer's use of the eDonkey network, since a low ID peer has no public IP to which other peers can connect to, thus all communication must be done through the eDonkey servers. This increases the computational overhead of servers, and results in reluctance of servers to accept low ID peers.

There are many programs that act as the client part of the network. Most notably, eDonkey2000, the original client by MetaMachine, closed-source but freeware, and no longer maintained but very popular in its day; and eMule [14, 15], a free program for Windows written in Visual C++ and licensed under the GNU GPL. eMule is in constant development and currently represents about 90% of all clients in the eD2k network.

BitTorrent Currently the most popular HM-based systems are those based on the BitTorrent protocol [8], that focuses on high data transfer speed rather than on search capabilities. In 2008 BitTorrent was the largest file sharing network (with 30% of the global peer-to-peer traffic, and 15% of the overall Internet traffic).

The specificity of BitTorrent is the notion of *torrent*, which defines a session of transfer of a single content to a set of peers. Torrents are independent, *i.e.*

participating in a torrent does not bring any benefit for the participation to another torrent.

A peer which wants to share a content (file or directory), creates a .torrent static metainfo file and publishes it to one of the torrent Web servers. Each server is responsible for linking the .torrent file with a suitable *tracker*, which keeps a global registry of all providers of the corresponding file. Trackers are responsible for helping downloaders find each other, to let them form groups of peers which are interested in the same files. Trackers are the only centralized components of BitTorrent, but they are not involved in the actual distribution of the files. They keep track of the peers currently involved in the torrents and collect statistics on the torrents. Trackers speak a very simple protocol layered on top of HTTP. Each downloader sends information about what file it is downloading, what port it is listening on, etc. and the tracker replies with a list of contact information for peers which are downloading the same file.

In each group, peers which have the complete file are called *seeds*, while peers which only have parts of the file and are trying to download the other parts are called *leechers*. The initial seed is the peer that is first source of the content.

A peer joins an existing torrent by downloading a related .torrent file from one of the torrent Web server. The .torrent file comes to the leecher along with the IP address of the tracker of the torrent. When joining the torrent, the peer asks to the tracker a list of IP address of peers to build its initial peer set. This list typically consists of 50 peers chosen at random in the list of peers currently involved in the torrent. The initial peer set will be augmented by peers connecting directly to this new peer. Such peers are aware of the new peer by receiving its IP from the tracker. Each peer reports its state to the tracker every 30 minutes in steady-state regime, or when disconnecting from the torrent, indicating each time the number of bytes it has uploaded and downloaded since it joined the torrent.

Each peer maintains a list of other peers it knows about. We call this list *peer set* (also known as neighbor set). A peer can only send data to a subset of its peer set. We call this subset *active peer set*. The choke algorithm, described below, determines the peers being part of the active peer set.

A torrent can thus be viewed as a collection of interconnected peer sets. By default, the maximum peer set size is 80. A peer should not exceed a threshold of 40 initiated connections among the 80 at each time. As a consequence, the 40 remaining connections should be initiated by remote peers. This policy guarantees a good interconnection among the peer sets in the torrent. If the set size of a peer set falls under 20, the peer will contact the tracker again to obtain a new list of IP addresses of peers.

Some peers limit the number of upload connections. Thus the existence of a a link between two nodes depends on many factors. To represent the topology of a group of downloaders, we should use an undirected graph. For example, in Figure 8 four cases are illustrated:

1. 1 seed for each file (groups 1,2,3)

2. 1 seed for n files, thus n possible groups (groups 1 and 4, groups 2 and 3)

3. m seeds for the same file (group 4)
4. 1 leecher for many files (across groups 1 and 2)

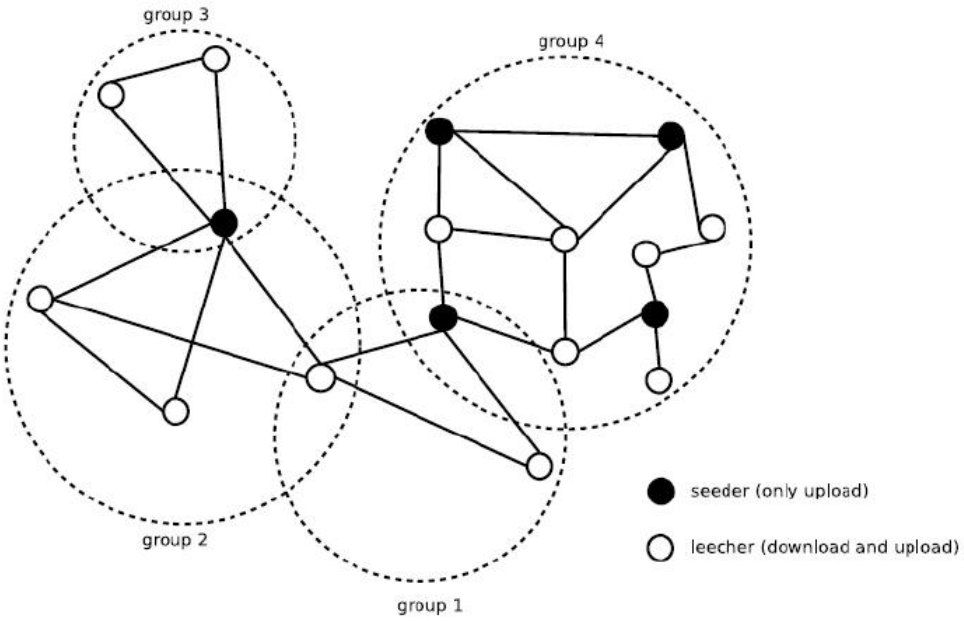


Figure 8: Different kinds of peer sets in BitTorrent.

Files transferred using BitTorrent are split into *pieces* (of 256KB), and each piece is split into *blocks* (of 16KB). Blocks are the transmission units on the network, but the protocol only accounts for transferred pieces. In particular, partially received pieces cannot be served by a peer, only complete pieces can. BitTorrent encrypts both the header and the payload, for each transferred piece. Using only 60-80 bits for the cipher, the aim is not to protect the data but instead to simply obfuscate the stream enough so that it is not detectable without incurring much of a performance hit. Although it is still possible to detect encrypted BitTorrent streams using sophisticated methods based on pattern and timing of the traffic, in practice, it is much harder to filter encrypted streams now. Encryption of P2P traffic seems to be picking up, as currently about 20% of BitTorrent traffic is encrypted.

Each downloader reports to all its peers what pieces it has, thus each peer knows the distribution of the pieces for each peer in its peer set. We say that peer B is *interested* in peer A when peer A has pieces that peer B does not have. We say that peer A *chokes* peer B when peer A decides not to send data to peer B. Conversely, peer A *unchokes* peer B when peer A decides to send data to peer B. Blocks can be downloaded from different peers. Requests for blocks are pipelined (typically 5), so delay between pieces being sent is avoided because every time a block arrives a new request is sent. Selecting pieces in good order is very important for good performance.

The *Rarest First Algorithm (RFA)* is the piece selection strategy used in BitTorrent. Pieces that have been already served at least once by the initial seed are called *available pieces*. The *rarest pieces* are the pieces that have the least number of copies in the peer set. In the case the least replicated piece in the peer set has m copies, then all the pieces with m copies form the *rarest pieces set*. Each peer maintains a list of the number of copies of each piece in its peer set. It uses this information to define a rarest pieces set. Let m be the number of copies of the rarest piece, then the index of each piece with m copies in the peer set is added to the rarest pieces set. The rarest pieces set of a peer is updated each time a copy of a piece is added to or removed from its peer set. Each peer selects the next piece to download at random in its rarest pieces set. There are three additional policies which can modify the behavior of the RFA: *random list policy*, *strict priority policy*, and *end game mode* [37].

The *Choke Algorithm (CA)* is the peer selection strategy used in BitTorrent, introduced to guarantee a reasonable level of upload and download reciprocation. As a consequence, free riders (*i.e.* peers that never upload) should be penalized. For the sake of clarity, *interested* always means interested in the local peer, and *choked* always means choked by the local peer. For a peer in leecher state, at most 4 peers can be unchoked by the peer and interested in the peer at the same time. Peers are unchoked using the following policy:

1. Every 10 seconds, the interested remote peers are ordered according to their download rate to the local peer and the 3 fastest peers are unchoked (we call them regular unchoked (RU) peers).

2. Every 30 seconds, one additional interested remote peer is unchoked at random. We call this random unchoke the optimistic unchoke (OU). The OU has two purposes. It allows to evaluate the download capacity of new peers in the peer set, and it allows to bootstrap new peers that do not have any piece to share by giving them their first piece.

In versions of the BitTorrent protocol previous to 4.0.0, the choke algorithm was the same in leecher state and in seed state. In the following we describe the new algorithm for the seed state, with which peers are no longer unchoked according to their upload rate from the local peer, but according to the time of their last unchoke. For a peer in seed state, at most 4 remote peers can be unchoked and interested at the same time. Peers are unchoked using the following policy:

1. Every 10 seconds, the unchoked and interested remote peers are ordered according to the time they were last unchoked, most recently unchoked peers first.

2. For two consecutive periods of 10 seconds, the 3 first peers are kept unchoked and an additional 4^{th} peer that is choked and interested is selected at random and unchoked.

3. For the third period of 10 seconds, the 4 first peers are kept unchoked.

Gnutella The Gnutella DUM-based protocol [23] was published in late 1999 by Nullsoft (inventor of WinAmp media player), and now it is at the 4th release. The protocol defines the way in which peers communicate over the Gnutella overlay network, that is through a set of descriptors whose inter-peer exchange is governed by a set of rules.

A Gnutella node connects itself to the network by establishing a connection with another node, typically one of the several known hosts that are almost always available. In general, the acquisition of another peer's address is not part of the protocol definition.

The messages allowed in the Gnutella network can be grouped as follows: Group Membership (PING and PONG descriptors, for peer discovery queries/replies), Search (QUERY and QUERY HIT descriptors, for file discovery queries/replies), and File Transfer (GET and PUSH descriptors, for file exchange between peers). Every PING or QUERY message received by a node is forwarded to all the neighbors of the node (flooding). The specification makes no recommendations as to the frequency at which a peer should send PING descriptors, although peer developers should make every attempt to minimize PING traffic on the network. To avoid network congestion, the PING and QUERY messages are always associated to a Time To Live (TTL), which is the max number of times the descriptor can be forwarded before it is removed from the network. The number of times the descriptor has been forwarded is named Hops.

$$TTL(0) = TTL(i) + Hops(i) \tag{5}$$

PONG and QUERY HIT descriptors may only be sent along the same path that carried the incoming PING and QUERY messages. A peer receiving a descriptor with the same Payload Descriptor and Descriptor ID as one it has received before, should attempt to avoid forwarding the descriptor to any connected peer. Many peer applications based on the Gnutella protocol are available for Windows, Linux and Mac OS X; the most famous is LimeWire [43].

Traffic measurements ([63], [64]) have demonstrated that Gnutella's overlay network topology does not match the underlying Internet topology. This leads to ineffective use of the physical networking infrastructure and to some lack of robustness, but also to quite good fault tolerance characteristics.

Moreover, various studies show that the distribution of Gnutella queries is similar to the distribution of HTTP requests in the Internet. They both follow the Zipf's law (the relative popularity of the jth resource is $q_j \propto 1/j^{\tau}$, with τ close to 1). Therefore, the proxy cache mechanism used in the Web context might have useful applications in the Gnutella P2P context. The difference between Gnutella caching and Web caching is in *content* location. In traditional Web caching, the content is provided by proxies which are well-defined Web servers. On the contrary, in Gnutella, the content is the sum of the responses to a query and so it is provided by several nodes. Moreover, Gnutella caching systems need to take into account not only the text of the query request but also the query's TTL, the neighbor that issued the request, and in general all the factors that define the coverage of a query's response. In the approach to Gnutella caching proposed by Markatos [49], when peer n receives a

query from neighbor n_2, it checks its cache to see if a query with the same text and the same TTL has been seen in the past. If such a query is found, and if this query has been sent in the past by neighbor n_2, then n returns the responses that exist in his cache for this query. If, on the other hand, n finds such a query in its cache, but the (cached) query had been sent by neighbor n_1, n forwards the (new) query to n_1, receives the results, combines them with the locally cached results of the query, and sends the combined results to n_2.

MUTE The MUTE network [51] uses Utility Counters (UCs) instead of TTLs. UCs are a simple mechanism that limits how far a query travels based on both the branching factor at each query hop (*i.e.* number of neighbor a message is forwarded to) and the number of results generated by the query. UCs have all the properties described in the previous section, and they can completely replace TTLs.

Each search query is tagged with a UC, and the UC starts out at 0. Search queries are forwarded with a broadcast scheme just like they are in a TTL network, and the UC on a query is modified before forwarding in two different ways. First, if a node generates results for a given query, the node adds the number of results generated to the UC before forwarding the query. Second, the number of neighbors that a node plans to forward the query to is added to the UC. For example, if a node receives a query with a UC of 25, and it generates 10 results, then it first increases the UC to 35. If it plans to forward the query to 6 of its neighbors, it increases the UC again to 41, and then it forwards a query to its neighbors.

Nodes in a UC-based network each enforce a UC limit: if a search query's UC hits this limit, it is dropped without being forwarded to neighbors. For example, if the UC limit is 100, a node that drops a query can surmise one of the following: that the query has generated at least 100 results, that the query has passed through hops with a total branching factor of 100, or that the query has reached the limit through a combination of both factors (for example, that it has generated 75 results and has passed through hops with a total branching factor of 25). Thus, the UC limit approximates an accurate control over how much utility a search sender can glean from the network with a single query.

Freenet The Freenet DUM-based P2P system [12, 20] was conceptualized by Clarke in 1999, and public development of the open source reference implementation began in early 2000. Freenet is a distributed information storage system which represents a prime example of how anonymity can be built into a P2P application. In designing Freenet, the authors focused on: privacy for information producers, consumers and holders; resistance to information censorship; high availability and reliability through decentralization; and efficient, scalable, and adaptive storage routing. Freenet uses a forwarding scheme for messages to ensure that the original requestor of a service cannot be tracked. It increases anonymity by using probabilistic algorithms so that origins cannot be easily tracked by analyzing network traffic.

Freenet participants provide storage space. A set of files is maintained locally up to the maximum disk space allocated by the node operator. When all disk

space is consumed, files are replaced in accordance with a least recently used (LRU) replacement strategy.

To each file to be shared, a location-independent globally unique identifier (GUID) is assigned. Freenet GUIDs are calculated is generated by hashing (with SHA-1) the contents of the file to be stored. To add a new file, a user sends the network an insert message containing the file and its assigned GUID, which causes the file to be stored on some set of nodes.

During a file's lifetime, it might migrate to or be replicated on other nodes. To retrieve a file, a user sends out a request message containing the GUID. When the request reaches one of the nodes where the file is stored, that node passes the data back to the request's originator.

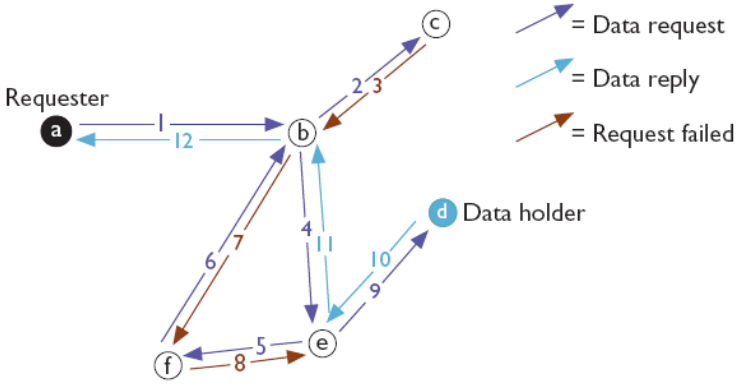


Figure 9: Typical request sequence.The request moves through the network from node to node, backing out of a dead-end (step 3) and a loop (step 7) before locating the desired file.

Every node maintains a routing table that lists the addresses of other nodes and the files it is supposed they hold. Freenet nodes pass and forward messages to find the location of an existing file or the proper location to store a new file. Freenet attempts to cluster files with similar keys on a single node. When a node receives a query, it first checks its own store, and if finds the file, returns it with a tag identifying itself as the data holder. Otherwise, the node forwards the request to the node in its table with the closest key to the one requested. That node then checks its store, and so on. If the request is successful, each node in the chain passes the file back upstream and creates a new entry in its routing table associating the data holder with the requested key. Depending on the distance from the holder, each node might also cache a copy locally.

Simulations [47] demonstrated Freenet's good scalability and fault tolerance. According to Kleinberg [35], Freenet should find data in around $O(log_2 n)$ hops, if peers are connected in a small-world network. However, this is only true on a mature Freenet network, and the typical churn rate, due to nodes not running continually and people trying it out and then leaving, may prevent it. Apart from this, the price to pay for anonymity, *i.e.* forwarding data chunks through several peers rather than directly from source to destination, is a high transfer overhead which can be accepted only for a limited set of applications.

Freenet protocol has other drawbacks. First of all, it is not possible to distinguish between a slow Freenet node sitting at the end of a slow modem line in the remote Australian outback, and a powerful node connected to a T3 in downtown Los Angeles. Moreover, the only real way to test improvements to the protocol is to see how they affect a large scale network, either in simulation, or in the real world - this leads to a slow and cumbersome development cycle. Clarke et al. are developing the Next Generation Routing, which aims to make Freenet nodes much smarter about deciding where to route information. For each node reference in its routing table, a node will collect extensive statistical information including response times for requesting particular keys, the proportion of requests which succeed in finding information, and the time required to establish a connection in the first place. When a new request is received, this information is used to estimate which node is likely to be able to retrieve the data in the least amount of time, and that is the node to which the request is forwarded.

DKS In academics, Distributed K-ary Search (DKS) [57] is probably the most known DSM-based protocol, including Chord [70] and Pastry [65]. Each instance of DKS is a fully decentralized overlay network characterized by three parameters: N the maximum number of nodes that can be in the network, k the search arity within the network, and f the degree of fault tolerance. Once these parameters are instantiated, the resulting network has several desirable properties, *e.g.* each lookup request is resolved in at most $\log_k(N)$ overlay hops. Each node has to maintain only $(k-1)\log_k(N)+1$ addresses of other nodes for routing purposes.

Chord [70] is the most famous DKS(N, 2, f) protocol. Given a key (*i.e.* the identifier of a resource or a service), the protocol maps the key onto a node (a host or a process identified by an IP address and a port number). Chord's consistent hash function assigns each node and key an m-bit identifier using a base hash function such as SHA-1. A node's identifier is chosen by hashing the node's IP address, while a key identifier is produced by hashing the key. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible. Identifiers are ordered on an *identifier circle* modulo 2^m. Key k is assigned to the first node whose identifier is equal or follows the identifier of k. This node is called the *successor node* of key k. Figure 10 shows a Chord ring with $m = 4$.

Key location, according to the Chord basic lookup algorithm, requires that each node knows how to contact its current successor node on the identifier circle. Queries for a given key identifier are passed around the circle via these successor pointers until they encounter a pair of nodes that straddle the desired identifier; the second in the pair is the node the query maps to. Figure 11 shows an example in which node $N1$ performs a query for key $K9$. The result returns along the reverse of the path followed by the query.

Chord's basic lookup algorithm, illustrated above, uses a number of messages which is linear in the number of nodes. To accelerate lookups, each node n could maintain a routing table with up to m entries, called the *finger table*. The i^{th} entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i-1} on the identifier circle; i.e. $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$

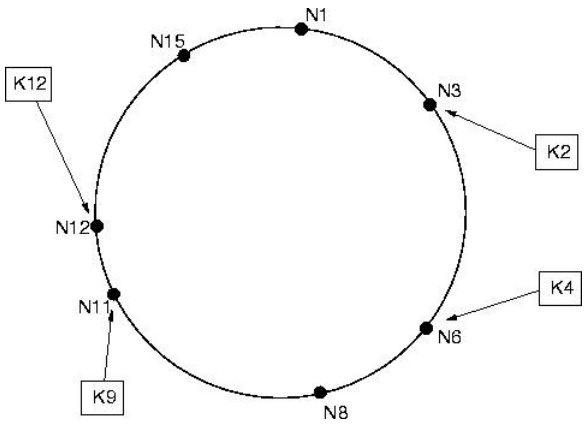


Figure 10: An identifier circle consisting of 7 nodes storing 4 keys. The successor of identifier 2 is the node with identifier 3, so $K2$ is located at $N3$. Similarly for the other 3 keys.

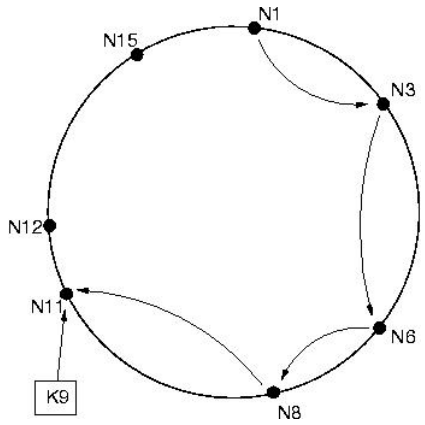


Figure 11: The path taken by a query from $N1$, searching for $K9$.

and all the arithmetic is module 2^m. We call node s the i^{th} *finger* of node n, and denote by $n.finger[i]$. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant node. Figure 12 illustrates the scalable lookup algorithm based on finger tables. In general, if node n searches for a key whose ID falls between n and its successor, node n finds the key in its successor; otherwise, n searches its finger table for the node n' whose ID most immediately precedes the one of the desired key, and then the basic algorithm is executed starting from n'. It is demonstrated that, with high probability, the number of nodes that must be contacted to find a successor in an N-node network is $O(\log N)$.

In order to ensure that lookups execute correctly as the set of participating nodes changes, Chord introduces a stabilization protocol that each node should run periodically in the background and which updates finger tables and successor pointers. If any sequence of join operations is executed interleaved with stabilizations, then at some time after the last join the successor pointers will form a cycle on all the nodes in the network. In other words, after some time each node is able to reach

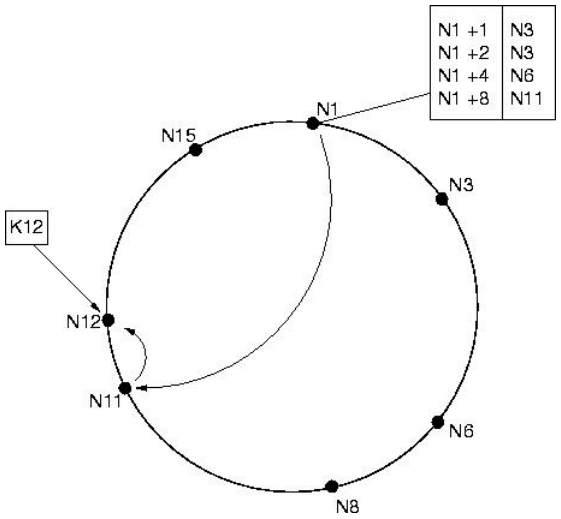


Figure 12: The finger table entries for node $N1$ and the path taken by a query from $N1$, searching for $K9$ using the scalable lookup algorithm.

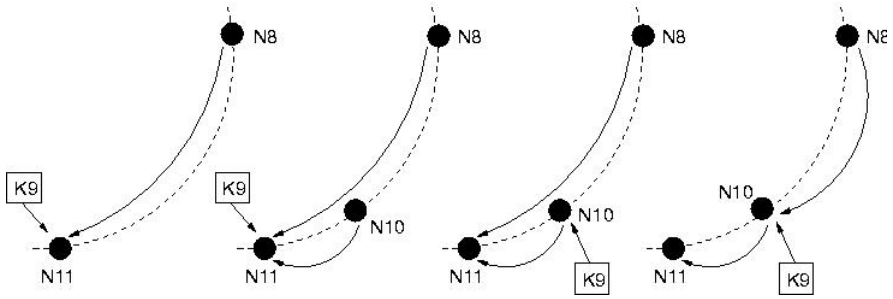


Figure 13: Example illustrating the join operation, including the stabilization procedure.

any other node in the network by following successor pointers. Moreover, if we take a stable network with N nodes (stable means that all successor and finger pointers are correct), and another set of up to N nodes joins the network, then lookups will still take $O(\log N)$ time with high probability.

The robustness of the Chord protocol relies on the fact that each node knows its successor. However, this assumption can be compromised if nodes fail. To increase robustness, each Chord node maintains a *successor list* of size r, containing the node's first r successors. If a node's immediate successor does not respond, the node can query the second entry in its successor list. If the length of the successor list is $r = \Omega(\log N)$, in a network that is initially stable, and then every node fails with probability $1/2$, then with high probability the procedure to find a successor returns the closest living successor.

A node that voluntarily leaves the system has to be treated as a node failure. To improve performance, a node n that is about to leave may transfer its keys to its successor and it may notify its predecessor p and its successor s. In turn, node p substitutes n with s in its successor list, and s replaces its predecessor with p.

Pastry [65] is a DKS(N, 2^b, f) protocol; b is a configuration parameter with typical value 4; keys have l digits of b bits, with typical value 16 or 32 (thus keys are 64 or 128-bits long). In each routing step, the message is forwarded to a node whose ID shares with the message key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current node's ID. Assuming a network consisting of N nodes, Pastry can route to the numerically closest node to a given key in less than $\lceil \log_{2^b} N \rceil$ steps under normal operation. Despite concurrent node failures, eventual delivery is guaranteed unless $\lfloor l/2 \rfloor$ nodes with adjacent node ID fail simultaneously.

Each Pastry node maintains a routing table R, organized into $\lceil \log_{2^b} N \rceil$ rows with $2^b - 1$ entries each. The choice of b involves a trade-off between the size of the populated portion of the routing table, approximately $\lceil \log_{2^b} N \rceil \times (2^b - 1)$ entries, and the maximum number of hops required to route between any pair of nodes $\lceil \log_{2^b} N \rceil$. With a value of $b = 4$ and 10^6 nodes, a routing table contains on average 75 entries and the expected number of routing hops is 5, whilst with 10^9 nodes the routing table contains on average 105 entries, and the expected number of routing hops is 7.

Pastry's protocol also defines mechanisms for handling the arrival and departure of nodes, and assumes that the application provides a function that allows each node to determine the distance of another node to itself (based on a scalar proximity metric, such as the number of IP routing hops or geographic distance), thus entries in the routing table of each node are chosen to be close to the current node.

Kademlia Kademlia [50] is another famous DSM-based protocol, used for example in recent versions of the eMule client (as an alternative to the traditional eDonkey protocol). Many of Kademlia's benefits result from its use of the XOR metric for distance between points in the key space. XOR is symmetric, allowing Kademlia participants to receive lookup queries from precisely the same distribution of nodes contained in their routing tables. To locate nodes near a particular ID, Kademlia uses a single routing algorithm from start to finish. In contrast, other systems use one algorithm to get near the target ID and another for the last few hops. Kademlia furthermore introduces a concurrency parameter that lets peers trade a constant factor in bandwidth for asynchronous lowest latency hop selection and delay-free fault recovery. Finally, Kademlia is the first peer-to-peer system to exploit the fact that node failures are inversely related to uptime.

FastTrack FastTrack [18] is the LM-based protocol used by file sharing applications such as KaZaA Media Desktop (shortly, KaZaA) [34] and iMesh [27]. Moreover, it is used by Skype [67], which is a VoIP system.

As of early 2003, FastTrack was the most popular file sharing network, being mainly used for the exchange of music MP3 files and having more users than Napster at its peak. Popular features of FastTrack are the ability to resume interrupted downloads and to simultaneously download segments of one file from multiple peers.

FastTrack extends the Gnutella protocol with supernodes, to improve scalability. Supernodes act as temporary indexing servers and help support the stability of the network. These supernodes stay outside the control of any company. A peer

sunning on a powerful computer with a fast network connection runs the software, will automatically become a supernode, effectively acting as a temporary indexing server for other, slower leaf peers. In order to be able to initially connect to the network, a list of supernode IP numbers is hardcoded in the peer. As soon as a peer finds a working supernode, it requests a list of currently active supernodes, to be used for future connection attempts. Each leaf peer uploads a list of files it intends to share to its supernode. It also sends search requests to this supernode. Supernodes communicate between each others in order to satisfy search requests. Peers are directly connected to exchange files; this transfer is based on the HTTP protocol. Programmers from the open source community have reverse engineered the portion of the protocol dealing with leaf-supernode communication, but the supernode-supernode communication protocol remains largely unknown. Examples of FastTrack open source implementations are:

- giFT-FastTrack, which is a giFT plugin allowing to search and download files on the FastTrack network;
- MLDonkey, which is a free multi-platform multi-network file sharing tool which is able to connect to the FastTrack network.

FastTrack was created in March 2001 by programmers of the Dutch company Kazaa BV, which used it to create a file sharing network called KaZaA. This was shortly before Napster was shut down by lawsuits. The large boost in KaZaA's population can be partially attributed to the impressive userbase of MusicCity, whose users in April 2001, discovered that their OpenNap network had closed down and were moved over to the Morpheus program which used the FastTrack network. Kazaa BV lost a lawsuit in the Netherlands in November 2001, and subsequently transferred ownership to a complicated mesh of offshore companies, with Sharman Networks being the most prominent one. In March 2002, a Dutch appeals court ruled that KaZaA was legal however, as the owners of the network were not responsible for possibly infringing actions of the users. In late February 2002, Morpheus was shut out by a protocol change, as the owners of Morpheus had failed to pay license fees to the owners of the FastTrack technology.

In April 2002, it was revealed that KaZaA also connected to another private network called Altnet, with undesirable threats (such as spywares). Because of these sometimes hidden features, CNet's Download.com site stopped the distribution of the KaZaA tool. In the same month, a free unauthorized third-party version called KaZaA Lite was released, with these features being removed. KaZaA Lite (also called K-Lite) is a modified version of the KaZaA Media Desktop application which excludes adware and spyware and provides slightly extended functionality. It connects to the same FastTrack network and thus allows to exchange files with all KaZaA users; it can be downloaded free. It was created by third party programmers by modifying the binary of the original KaZaA application. Included with recent versions of KaZaA Lite is K++, a memory patcher that removes search limit restrictions, multisource limits, and sets one's "participation level" to the maximum of 1000. The K-Lite application is available on the FastTrack network and can be downloaded with KaZaA.

Sharman Networks considers KaZaA Lite to be a copyright violation. On August 2003, Sharman sent a letter to Google requesting that all links to KaZaA Lite be removed from their database, as they consider KaZaA Lite to be a copyright violation. During December 2003 Sharman Networks emailed the hosts of all sites with a copy of K++, threatening legal action if it was not removed. Therefore it is difficult to get and is no longer on the official web site. There are still some web sites on which it is available, and it is widely available on the KaZaA Network itself.

Gnutella2 The Gnutella2 network [24] is a self-organizing collection of interconnected nodes cooperating to enable productive distributed activities. According to the Layered Model, not all of the nodes participating in the system are equal: there are two primary node types, "hubs" and "leaves". The goal is to maximize the number of leaves and minimize the number of hubs, however, due to the limited nature of resources the maximum viable ratio of leaves to hubs is limited. This quantity is known as the "leaf density".

Leaf nodes are the most common node type on the network - they have no special responsibilities and do not form a working part of the network infrastructure. Nodes with limited resources must operate as leaf nodes: this includes limited bandwidth, CPU or RAM, low or unpredictable expected uptime, and inability to accept inbound TCP or UDP.

Hub nodes on the other hand form an important and active part of the network infrastructure, organizing surrounding nodes, filtering and directing traffic over several media types. Hub nodes devote substantial resources to the network, and as a result their capacity to participate in higher level network functions is limited. Only the most capable nodes are selected to act as hubs. Nodes operating as Gnutella2 hubs have a set of responsibilities to meet. Hubs are highly interconnected, forming a "hub network" or "hub web", with each hub maintaining a connection to 5-30 other "neighboring" hubs. The number of hub interconnections must scale up with the overall size of the network. Each hub also accepts connections from a large collection of leaf nodes, typically 200-300 depending on available resources. Leaf nodes are considered to be the "edge" of the network. In practice, leaves simultaneously connect to two hubs, however, from the point of view of the hubs each leaf is considered a dead end. The goal is to maximize the number of leaves and minimize the number of hubs, however, due to the limited nature of resources the maximum viable ratio of leaves to hubs is limited. This quantity is known as the "leaf density".

The Gnutella2 resource search mechanism can be described as an "iterative crawl" of the network, with a series of important optimizations derived from the network topology and components. A search peer iteratively contacts known hubs, that match the query against the cached hash tables of their neighbors. If a table hit occurs, the query is forwarded once only. Nodes which actually receive the filtered query process it and send results to the search peer directly.

Globally unique identifiers (GUIDs) are used to identify nodes on the network. Each Gnutella2 node must maintain a dynamic node route cache to map node GUIDs to appropriate destinations. The route cache is consulted when a packet needs to be dispatched to, or toward a GUID-addressed node. GUID-addressing is used over network-addressing in a number of situations.

In the Gnutella2 network architecture, neighboring nodes exchange compressed hash filter tables describing the content, which can be reached by communicating with them. These tables are updated dynamically as necessary. Query Hash Tables (QHTs) provide enough information to know with certainty that a particular node (and possibly its descendants) will not be able to provide any matching objects for a given query. Conversely, the QHT may reveal that a node or its descendants may be able to provide matching objects. The Gnutella2 object search mechanism is best described as an "iterative crawl" of the network, with a series of important optimizations derived from the network topology and components:

- a search client iteratively contacts known hubs with its query;
- hubs match the query against the cached hash tables of their neighbors;
- where a table hit occurs, the query is forwarded once only;
- nodes which actually receive the filtered query process it and send results to the search client directly.

6.2 Distributed Storage

Tedeschi, et al. [73] propose a "policy architecture" to develop distributed storage systems, which divides the control and data plane, where the control plane embodies the design policy of the system over a data plane that provides a set of common mechanisms. Using such an approach, the authors build a distributed storage systems spanning a large portion of the design space. A popular distributed storage systems based on the Layered Model (LM) is Wuala [81], which uses a Chord-like overlay scheme to organize a group of supernodes, each one managing a cluster of storage nodes.

In the same context, Lv et al. [46] experimentally showed that replication improves the search performance of DUM-based P2P networks (see also Section 2). To optimize network-wide search performance given limited storage capacity, more replicas are preferred for more frequently accessed objects. Cohen and Shenker [13] proved that the search time and traffic under random walk search is minimized when the number of replicas for each object is proportional to the square root of its query rate. Tewari and Kleinrock [74] showed that under controlled flooding search, the search traffic is minimized under the same square-root replica distribution, whereas the search time is minimized when the number of replicas for each object is linearly proportional to its query rate. Kangasharju, et al. [33] developed a logarithmic replication distribution to maximize the content availability under intermittent connectivity. However, none of the above work has considered keeping the replicas consistent with the authoritative contents. In general, there are two classes of methods to maintain consistency: push-based and pull-based. In push-based methods, the content owners keep track of the replica locations and send invalidation messages or updated contents to the replicas whenever the contents are modified. In contrast, pull-based methods are replica-driven. The replicas, when considered outdated, are validated before serving new requests. Tang et al. [72] propose to assign each replica an expiration time (time-to-live, TTL) beyond which the replica stops

serving new requests unless it is validated. While TTL-based consistency is widely explored in many client-server applications, there has been no study on TTL-based consistency in P2P networks. The main contribution of [72] is an analytical model that studies the search performance and the freshness of P2P content sharing under TTL-based consistency. Due to the random nature of request routing, P2P networks are fundamentally different from most existing TTL-based systems in that every node with a valid replica has the potential to serve any other node. The authors identify and discuss the factors that affect the performance of P2P content sharing under TTL-based consistency. Simulation results indicate a trade-off between search performance and freshness: the search cost sub-linearly falls off with decreasing freshness of P2P content sharing.

6.3 Parallel and Distributed Computing

Parallel and distributed applications based on the P2P paradigm split a large task into smaller sub-pieces that can execute in parallel over a number of independent peer nodes. Most often, the same task is performed on each peer using a different set of parameters (compute-intensive applications). Example tasks are code breaking, market evaluation, scientific simulations.

Recently, the Grid computing research community started using the P2P paradigm for accomplishing efficient resource discovery, which is one of the fundamental requirements of Grid systems [62]. Resource discovery involves searching for resources that match the user's application requirements. Various kinds of solutions to Grid resource discovery have been developed, including the centralized and hierarchical information server approach. However, these approaches have serious limitations in regards to scalability, fault-tolerance and network congestion, with respect to solutions based on P2P. An almost complete survey on peer-to-peer resource sharing models for Grid systems is proposed in a recent work of Trunfio, et al. [77].

P2P networks have also begun to attract attention from scientists in other disciplines, especially those that deal with large data sets such as bioinformatics. In such context, P2P networks can be used for example to run programs designed to identify known and new protein sequences using high-throughput methods. A P2P network provides methods for accessing distributed resources with minimal maintenance cost. It also provides scalable techniques to search through large amounts of resources scattered through the network. Furthermore, joining or leaving the network becomes a simple task. These properties of P2P networks make the technology an ideal candidate to implement search through proteomics laboratories. A proteomics laboratory acting as a peer in a P2P network will share its complete or partial data repository so that other peers and itself can benefit from it.

6.4 Blockchain

Bitcoin is well-known virtual currency, based on the blockchain technology [54]. A blockchain is a virtual ledger that stores all the transactions performed by users. There is a P2P network where each node stores a copy of the whole blockchain, in order to validate transactions and, possibly, to contribute to the growth of the

blockchain itself. This latter is a competetive process, where each node (denoted as miner) tries to add the most recent transactions to the blockchain (as a new block), by executing a randomized algorithm whose numerical result has to overcome a given threshold, in order to be accepted. The winner, *i.e.*, the miner that solves the challenge first, spreads the block within the P2P network and all other nodes add it to their local blockchain copy. Only mined transactions are valid, and not effaceable. Moreover, if the blockchain is public, any participant and in general anyone (provided with a suitable client software) can read the full history of transactions. To forge the blockchain, half participants plus one should make a coalition, which is highly unlikely.

Importantly, the blockchain is not only a platform for virtual currency. For example, Ethereum [16] can store *smart contracts*. According to the classical definition [71], a smart contract is a computerized transaction procotol that executes the terms of a contract. In the blockchain domain, a smart contract is a script that is stored in the ledger, provided with a unique identifier. The smart contract performs any kind of operations based on received input (transactions).

6.5 VoIP and Multimedia Streaming

Nowadays, traditional telephony is being flanked by Voice over IP (VoIP), which is a family of transmission technologies for delivery of voice communications over the Internet or other packet-switched networks. Skype, one of the most widely used Internet phone applications, is based on the FastTrack protocol (originally developed for the KaZaA file sharing network) [18]. Furthermore, many research organizations are trying to apply P2P networks to cellular networks.

Another kind of Internet-scale multimedia application which is getting enormous success is live P2P audio/video streaming (P2PTV), which is usually based on the Decentralized Unstructured Model. TVU is an Internet TV broadcasting network that uses P2PTV technology to offer its broadcasters global reach and low costs [79]. Joost is another known P2PTV Internet application [31]. The peer-to-peer layer comes from the Joltid company, which consists of the original management and development team behind KaZaA and the FastTrack peer-to-peer network, and also provided the peer-to-peer layer of Skype. TVants is a P2PTV software, whose core technology is based mainly on the BitTorrent protocol [8], for which a single peer playing the stream exchanges (sends or receives) streaming data packets with a few peers, in order to maximize the bandwidth usage and obtain the most optimal streaming quality [78]. The TVants peer allows users to watch 600 different channels from China or foreign countries, and to share their hard disk video local files with anyone online with broadband connection. LiveStation is a P2PTV platform being developed by Skinkers Ltd. based on peer-to-peer technology acquired from Microsoft Research [45]. Social networking features have been added that include the ability to chat with other viewers and also find out what others are watching through a user generated rating system.

Since P2PTV targets a lot of people, it needs group communication functionalities to be implemented in the hosts, on the edge of the network. Protocols for live P2P streaming application can be classified in three categories:

- source-driven
- receiver-driven
- data-driven

In the source-driven approach, there are a data plan, which is used to send data to all peers, and a control plan, which is used to manage the group and dynamism of peers (arrivals and departures). With Peercast [6], the control plan is a tree whose root is the source of the stream, and the data plan uses the same tree. Other control plans like that of ZigZag [75] defines hierarchical organization of peers into a cluster, with the data plan being again a tree built on top of the control plan.

With receiver-driven approach, control plan and data plan are clearly separated similarly to source-driven approach, and control plan can be a tree, a cluster or a mesh. Conversely, data plan is a tree and it is rooted at the receiver side instead of the source side. receiver-driven approach is usually related to data encoding like layered coding or multiple description coding as in Peerstreaming [39].

Unlike other solutions, the data-driven approach does not clearly separate control plan and data plan. Group members (peers) exchange control messages about data availability in the network. Each peer chooses its neighborhood according to the data it needs, searching for example by means of epidemic algorithms [17], like in CoolStreaming [83], PULSE [60] and PRIME [48].

6.6 Gaming

Massively multi-player games (MMGs) are becoming popular nowadays. In MMGs, a huge number of players are in the same virtual world, where they play roles, move and interact with each other and with other objects in the world. To overcome the limited scalability and robustness problems inherent to centralized solutions, P2P overlay networks are emerging as a promising architecture for MMGs [10]. A P2P overlay network can be easily applicable to MMGs since players are likely to contribute their communication and computation resources for interesting gameplay.

The authors in [7] describe interactive action games in which bandwidth demand is reduced by estimating what players are paying attention to, and the resource and interest heterogeneity are managed by disseminating updates via a multicast system designed for the special requirements of games.

6.7 Education and Academia

Due to the fast distribution and large storage space features, many organizations are trying to apply P2P networks for educational and academic purposes. For instance, Pennsylvania State University, MIT and Simon Fraser University are carrying on a project called LionShare designed for facilitating file sharing among educational institutions globally [44]. The LionShare project is dedicated to harnessing the promise of P2P file sharing and the integration of P2P with organizational services to create a collaborative environment for use in academic communities. The LionShare Peer

application is built around the themes of collaboration, security, personal responsibility, and access control of shared resources, along with access to large digital repositories. LionShare Peers operate within a P2P network using code originating from the Limewire 4.0 open source project. Limewire uses the Gnutella protocol for the basic search and retrieval functions.

Edutella is a peer-to-peer network for exchanging information about learning objects, rather than files [55]. It is built with semantic web technology applying the latest P2P research. Building blocks of Edutella are JXTA (see Section 5) and RDF, which is a W3C framework for representing resources in the Web.

6.8 Internet of Things

Smart devices that are able to communicate, provide useful data and accept commands are the building blocks of the Internet of Things (IoT) [80], an emerging domain where the P2P paradigm plays a pivotal role. IoT devices are usually wireless and powered by small batteries. They are constrained in terms of computing, storage and transmission capabilities. Thus, they require highly efficient algorithms and protocols.

7 Adaptivity in Peer-to-Peer Systems

The behavior of a peer is determined by its internal structure (that may have different configurations). Such a structure can be static, *i.e.* defined by fixed policies shared by every peer (protocols), or dynamically adjusted by an adaptive plan τ which determines successive restructuring (or reconfigurations, at least) in response to the environment.

In the last half-decade, many considerable peer-to-peer protocols have been proposed. They can be grouped in few architectural models, taking into account basically two dimensions: the dispersion degree of information about shared resources, and their logical organization. The behavior of a peer-to-peer system based on protocols (fig. 14) follows a pre-established pattern. *E.g.*, in Chord [70] messages are routed among peers according to a deterministic rule which considers the metric distance between message identifiers and peer identifiers.

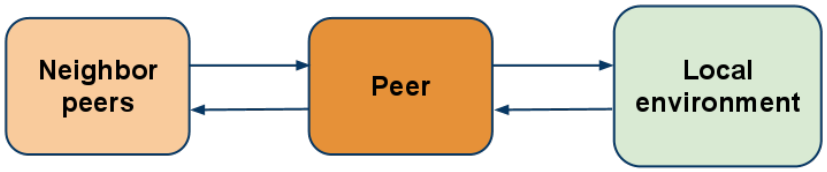


Figure 14: Interactions between a peer, its local environment and its neighbors in a traditional P2P architecture.

On the other side, there is a lack of common understanding about adaptive mechanisms. Traditional peer-to-peer networks are not are able to self-design, self-configure, self-monitor, self-deploy, self-adapt and self-heal - in other words, peer-to-peer networks lack *autonomic* features. The presence of a shared adaptive plan turns the P2P network in a complex adaptive system (CAS) [26], that always tends to an equilibrium configuration with (hopefully) optimal performance - according to explicit or implicit quality of service requirements.

In our view, illustrated by figure 15, the internal structure (or configuration, if the structure cannot change) of each peer may change in order to adapt to the environment. For example, consider a search algorithm whose parameters' values change over time in a different way for each peer depending on local performance indicators. Evolution can be *phylogenetic*, for which memoryless transformations are applied, or *cultural or epigenetic*, which assumes learning and knowledge transmission. In general, adaptive peer-to-peer networks may emulate the ability of biological systems to cope with unforeseen scenarios, variations in the environment or presence of deviant peers.

It may be observed that adaptation mechanisms running in background introduce additional costs in terms of local resources consumed and message traffic within the network. However, even the most advanced protocols of non-adaptive peer-to-peer systems require costly periodic self-maintenance, such as stabilization of routing tables [2].

Moreover, traditional P2P systems do not account any of the dynamics and

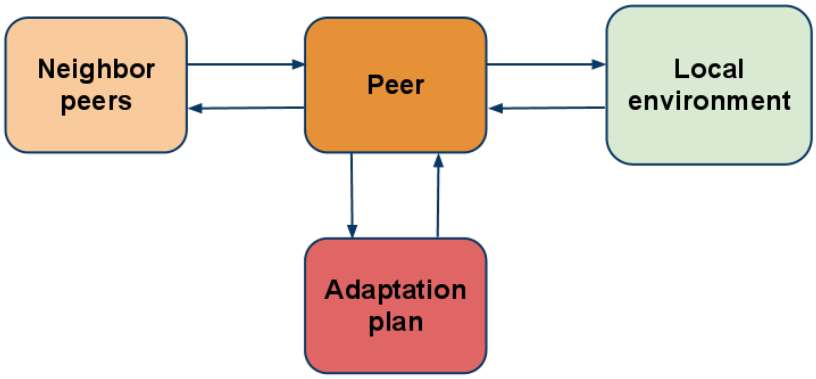


Figure 15: Interactions between a peer, local environment and neighbors in an adaptive P2P architecture.

heterogeneity of their environment. Systems like KaZaA and JXTA [2] do take heterogeneity of the running environment into account, by dividing peers in supernodes and regular nodes, and providing strategies for dynamic promotions and demotions. Such kind of adaptation strategy may be affected by too much control overhead if the processing power available to peers, which depends not only their hardware resources but also on usage load, quickly changes over time. Other systems incorporate IP network topology awareness into their design. An example is Pastry's proximity metric, for which each node ID match in the routing table is tagged with the distance, in terms of IP hops, of the current node from the given IP address [65]. The drawback of this approach is its limited scope. A more comprehensive strategy has been proposed by Kalyvianki *et al.* [32], where each node analyzes log traces in order to find node capacity as perceived by the P2P application, by processing raw events such as frequency of schedule changes, number of running processes and duration of file system operations. Resulting information is periodically published in the overlay network, in order to be used by P2P applications to select nodes with best overall performance. The limit of this approach is the high bandwidth consumption due to control messaging, which could be excessive in case of high dynamism in the availability of each node's resources.

References

[1] A. Abdul-Rahman, S. Halles, *A distributed trust model*. Proc. of the ACM Workshop on New Security Paradigms (NSPW '97), Langdale, Cumbria, U.K., September 1997.

[2] M. Amoretti, *A Survey of Peer-to-Peer Overlay Schemes: Effective- ness, Efficiency and Security. Recent Patents on Computer Science*, pp. 195-213, Vol. 2, Issue 3, ISSN 1874-4796, Ed. Bentham Science, November 2009.

[3] P. Antoniadis, B. Le Grand, *Incentives for resource sharing in selforganized communities: From economics to social psychology*. Proc. of the International Conference on Digital Information Management (ICDIM '07), Lyon, France 2007.

[4] E. Avatangelou, R.F. Dommarco, M. Klein, et al., *Conjoint PERSONA-SOPRANO Workshop*. Proc 1st European Conference on Ambient Intelligence (AmI-07), Darmstadt, Germany, November 2007.

[5] S.A. Baset, H. G. Schulzrinne, *An analysis of the skype peer-to-peer internet telephony protocol*, Proc. of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006), Barcelona, Spain April 2006.

[6] M. Bawa, H. Deshpande, H. Garcia-Molina, *Transience of peers & streaming media*. SIGCOMM Comput Commun Rev 2003; 33(1).

[7] A. Bharambe, J.R. Douceur, J.R. Lorch, et al, *Donnybrook: Enabling large-scale, high-speed, peer-to-peer games*. Proc. of the ACM SIGCOMM, Seattle, WA, USA, August 2008.

[8] BitTorrent official site. *http://www.bittorrent.org*.

[9] F. E. Bustamante, Y. Qiao. *Friendships that last: Peer lifespan and its role in P2P protocols*, Proc. of the International Workshop on Web Content Caching and Distribution, 2003.

[10] E. Buyukkaya, M. Abdallah, *Data management in voronoi-based P2P gaming*. Proc. of the 5th IEEE Consumer Communications and Networking Conf (CCNC 2008), 2008.

[11] C. M. Christensen, *The Innovator's Dilemma*, Harvard Business School Press, 1997.

[12] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, *Freenet: A distributed anonymous information storage and retrieval system*, Lecture Notes in Computer Science, 2001.

[13] E. Cohen, S. Shenker, *Replication strategies in unstructured peer-to-peer networks*. Proc. of the ACM SIGCOMM '02, Pittsburgh, PA, USA, August 2002.

[14] eMule official site. *http://www.emule-project.net*

[15] Y. Kulbak, D. Bickson, *The eMule Protocol Specification*, 2005.

[16] Ethereum official site. https://www.ethereum.org/

[17] P.T. Eugster, R. Guerraoui, A.M. Kermarrec, L. Massoulie, *From epidemics to distributed computing*, IEEE Computers, Vol.37, No.5, 2004.

[18] FastTrack documentation on WikiPedia. *http://en.wikipedia.org/wiki/FastTrack*

[19] Free Haven site. *http://freehaven.net/anonbib/date.html*

[20] Freenet official site. *http://freenet.sourceforge.net*

[21] C. GauthierDickey, C. Grothoff, *Bootstrapping of peer-to-peer networks*, Proc. of the International Workshop on Dependable and Sustainable Peer-to-Peer Systems, Turku, Finland, July 2008.

[22] D. Gambetta, *Can We Trust Trust? Trust: Making and breaking cooperative relations*, D. Gambetta (ed.), Basil Blackwell, Oxford 1990.

[23] Gnutella RFC, *http://rfc-gnutella.sourceforge.net*

[24] Gnutella 2 developer wiki, *http://g2.trillinux.org/index.php*

[25] Microsoft Office Groove official site, *http://office.microsoft.com/en-us/groove/default.aspx*

[26] F. Heylighen, *The Science of Self-organization and Adaptivity*, Knowledge Management, Organizational Intelligence and Learning, and Complexity, in The Encyclopedia of Life Support Systems (EOLSS), L. D. Kiel (ed.), Eolss Publishers, Oxford, 2001.

[27] iMesh homepage, *http://www.imesh.com*

[28] Ipoque GmbH. Internet Study 2007.

[29] IST Advisory Group. *Scenarios for ambient intelligence in 2010*. European Commission, 2001.

[30] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, H. Schulzrinne. *Resource location and discovery (reload) base protocol*. IETF Internet draft, December 2008.

[31] Joost homepage, *http://www.joost.com*

[32] E. Kalyvianki, I. Pratt, *Building Adaptive Peer-to-Peer Systems*, Proc. 4th IEEE Int'l Conference on Peer-to-Peer Computing (P2P'04), Zurich, Switzerland, 2004.

[33] J. Kangasharju, K. Ross, D. Turner, *Adaptive content management in structured P2P communities*. Proc. of the International ICST Conference on Scalable Information Systems (INFOSCALE '06), Hong Kong, China, June 2006.

[34] KaZaA official site. *http://www.kazaa.com*

[35] J. Kleinberg, *The Small-World Phenomenon: An Algorithmic Perspective*, Proc. of the 32nd ACM Symposium on Theory of Computing, 1999.

[36] M. Kleis, E.K. Lua, X. Zhou, *Hierarchical peer-to-peer networks using lightweight superpeer topologies*, Proc. of the 10th IEEE Symposium on Computers and Communication (ICSS05), La Manga del Mar Menor, Cartagena, Spain, June 2005.

[37] A. Legout, G. Urvoy-Keller, and P. Michiardi, *Rarest first and choke algorithms are enough*, Proc. of the 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeriro, Brazil, 2006.

[38] D. Leonard, Y. Zhongmei, V. Rai, D. Loguinov, *On Lifetime-Based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks*, IEEE/ACM Transactions on Networking, Vol. 15, No. 3, pp. 644-656, 2007.

[39] J. Li, *Peerstreaming: A practical receiver-driven peer-to-peer media streaming system*. Proc. of the 7th IEEE Workshop on Multimedia Signal Processing, Shanghai, China October 2005.

[40] B. Li, H. Yin, *Peer-to-peer live video streaming on the internet: Issues, existing approaches, and challenges*. IEEE Computer Magazine 2007, Vol.45, No.6.

[41] J. Liang, N. Naoumov, K. Ross. *The Index Poisoning Attack on P2P File-Sharing Systems*. Proc. of the IEEE INFOCOM, Barcelona, Spain, April 2006.

[42] Liben-Nowell, D. and Balakrishnan, H. and Karger, D., *Analysis of the evolution of peer-to-peer systems*, 21st ACM Annual Symposium on Principles of Distributed Computing, pp. 233-242, 2002.

[43] Limewire official site, *http://www.limewire.com*

[44] LionShare official site, http://lionshare.its.psu.edu

[45] LiveStation official site. *http://www.livestation.com*

[46] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, *Search and Replication in Unstructured Peer-to-Peer Networks*, 16th International Conference on Supercomputing, New York, 2002.

[47] J. Mache, M. Gilbert, J. Guchereau, J. Lesh, F. Ramli, M. Wilkinson, *Request Algorithms in Freenet-Style Peer-to-Peer Systems*, Proc. of the IEEE Second International Conference on Peer-to-Peer Computing (P2P '02), 2002.

[48] N. Magharei, A.H. Rasti, *Prime: Peer-to-peer receiver-driven meshbased streaming*. Proc. of the 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Anchorage, Alaska , USA, May 2007.

[49] E. Markatos, *Tracing a large-scale Peer-to-Peer System: an hour in the life of Gnutella*, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.

[50] P. Maymounkov, D. Mazieres, *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, 1st International Workshop on Peer-to-peer Systems, March 2002.

[51] MUTE official site. *http://mute-net.sourceforge.net*

[52] N. Naoumov, K. Ross, *Exploiting P2P systems for DDoS attacks*, Proc. of the International Workshop on Peer-to-Peer Information Management (P2PIM), Hong Kong, May 2006.

[53] Napster official site. *http://free.napster.com*

[54] A. Narayanan, J. Clark, *Bitcoin's Academic Pedigree*, ACM Queue, July-August 2017.

[55] W. Nejdl, B. Wolf, C. Qu, *EDUTELLA: A P2P networking infrastructure based on RDF*. Proc. of the ACM WWW2002, Honolulu, Hawaii, USA, May 2002.

[56] Netcraft, *P2P Networks Hijacked for DDoS Attacks*, 2007.

[57] L. Onana Alima, El-Ansary, S. and Brand, P. and Haridi, S., *DKS(N,k,f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Communications*, Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '03), 2003.

[58] OpenNap-NG official site. *http://opennap-ng.sourceforge.net*

[59] Peer-to-Peer Session Initiation Protocol (p2psip) working group. http://www.ietf.org/html.charters/p2psip-charter.html

[60] F. Pianese, D. Perino, J. Keller, E.W. Biersack, *PULSE: An Adaptive, incentive-based, unstructured P2P live streaming system*. IEEE Transactions on Multimedia, Vol.9, No.8, 2007.

[61] C. Ramos, J.C. Augusto, D. Shapiro, *Ambient Intelligence - the next step for artificial intelligence*. IEEE Intelligent Systems, Vol.23, No.2, 2008.

[62] R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, R. Buyya, *Decentralised resource discovery service for large scale federated grids*. Proc. of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Bangalore, India, December 2007.

[63] M. Ripeanu, *Peer-to-Peer Architecture Case Study: Gnutella Network*, Technical Report, University of Chicago, 2001.

[64] M. Ripeanu, I. Foster, A. Iamnitchi, *Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design*, IEEE Internet Computing Journal (special issue on peer-to-peer networking), Vol. 6, No. 1, 2002.

[65] A. Rowstron, P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, Proc. of IFIP/ACM Middleware, 2001.

[66] S. Saroiu, P. K. Gummadi, S. D. Gribble, *A Measurement Study of Peer-to-Peer File Sharing Systems*, Proc. of the Multimedia Computing and Networking (MMCN), San Jose, 2002.

[67] Skype homepage, *http://www.skype.com*

[68] Slashdot. *Comcast continues to block peer to peer traffic*. 2007. http://yro.slashdot.org/article.pl?sid=07/12/01/0011253

[69] Soulseek official site. *http://www.slsknet.org*

[70] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications*, IEEE/ACM Transactions on Networking, Vol.11, No.1, 2003.

[71] N. Szabo, *Smart Contracts: Building Blocks for Digital Markets*, http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[72] X. Tang, J. Xu, W.C. Lee, *Analysis of TTL-based consistency in unstructured peer-to-peer networks*. IEEE Transactions on Parallel and Distributed Systems 2008; 19(12).

[73] C. Tedeschi, F. Desprez, E. Caron, *Efficiency of tree-structured peerto- peer service discovery systems*. Proc. of HotP2P '08, Miami, FL, USA, April 2008.

[74] S. Tewari, L. Kleinrock, *Proportional replication in peer-to-peer networks*. Proc. of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), Barcelona, Spain, April 2006.

[75] D. Tran, K. Hua, T. Do, *Zigzag: An efficient peer-to-peer scheme for media streaming*. Proc. 22nd Annual Joint Conf of the IEEE Computer and Communications Societies (INFOCOM), San Francisco, USA March 2003.

[76] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Poyoul, B. Yeager, *Project JXTA 2.0 Super-Peer Virtual Network*, Technical Paper, Sun MicroSystems, 2003.

[77] P. Trunfio, D. Talia, H. Papadakis, et al. *Peer-to-peer resource discovery in grids: Models and systems*. Future Generation Computer Systems 2007; 23(7).

[78] TVants official site, *http://www.tvants.com*

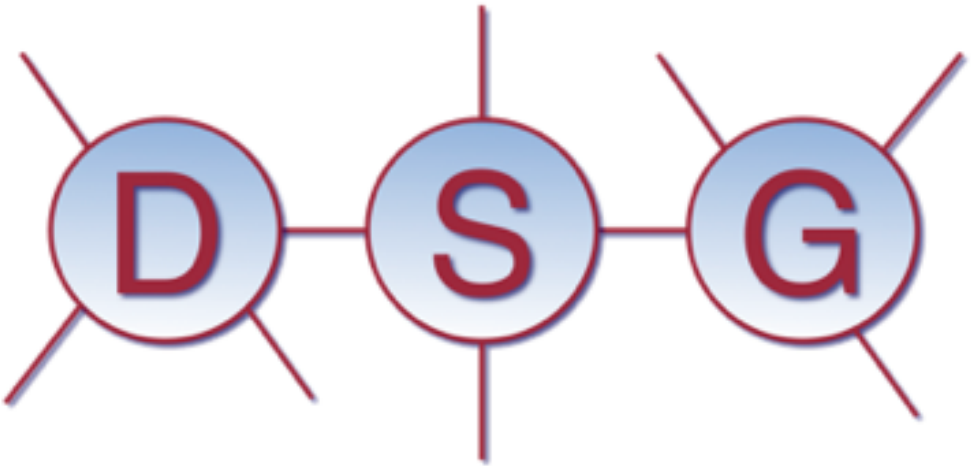
[79] TVU networks official site, *http://pages.tvunetworks.com*

[80] J.-P. Vasseur, A. Dunkels, *Interconnecting Smart Objects with IP*, Morgan Kaufmann, 2010.

[81] Wuala, the social online storage. http://www.wuala.com/en

[82] Z. Yao , D. Leonard , X. Wang , D. Loguinov, *Modeling heterogeneous user churn and local resilience of unstructured p2p networks*, Proc. of the 14th Int'l Conference on Network Protocols (ICNP '06), Santa Barbara, California, USA, November 2006.

[83] X. Zhang, J. Liu, B. Li, T.P. Yum, *CoolStreaming/DONet: A datadriven overlay network for peer-to-peer live media streaming*, Proc. of the 24th Annual Joint Conferences of the IEEE Computer and Communications Societies (INFOCOM), Miami, USA, March 2005.


D
S
G