P

Peer-to-Peer

Stefan Schmid[1], Roger Wattenhofer[2]
[1]Telekom Laboratories/TU Berlin, Berlin, Germany
[2]ETH Zürich, Zurich, Switzerland

Synonyms

Distributed hash table (DHT); Overlay network; Decentralization; Open distributed systems; Consistent hashing

Definition

The term *peer-to-peer* (p2p) is ambiguous, and is used in a variety of different contexts, such as:

- In popular media coverage, p2p is often synonymous to software or protocols that allow users to "share" files (music, software, books, movies, etc.). p2p file sharing is very popular and a large fraction of the total Internet traffic is due to p2p.
- In academia, the term p2p is used mostly in two ways. A narrow view essentially defines p2p as the "theory behind file-sharing protocols." In other words, how do Internet hosts need to be organized in order to deliver a search engine to find (share) content (files) efficiently? A popular term is "distributed hash table" (DHT), a distributed data structure that implements such a content search engine. A DHT should support at least a search (for a key) and an insert(key, object) operation. A DHT has many applications beyond file sharing, e.g., the Internet domain name system (DNS).
- A broader view generalizes p2p beyond file sharing: Indeed, there is a growing number of applications operating outside the juridical gray area, e.g., p2p Internet telephony à la Skype, p2p mass player games, p2p live audio&video streaming as in PPLive, StreamForge or Zattoo, or p2p social storage and cloud computing systems such as Wuala. Trying to account for the new applications beyond file sharing, one might define p2p as a large-scale distributed system that operates without a central server bottleneck. However, with this definition almost "everything decentralized" is p2p!
- From a different viewpoint, the term p2p may also be synonymous for privacy protection, as various p2p systems such as Freenet allow publishers of information to remain anonymous and uncensored.

In other words, there is no single well-fitting definition of p2p, as some definitions in use today are even contradictory. In the following, an academic viewpoint is assumed (second and third definition above).

Discussion

The Paradigm

At the heart of p2p computing lies the idea that each network participant serves both as a producer ("server") and consumer ("client") of services. Depending on the application, the shared resources can be data (files), CPU power, disk storage, or network bandwidth. Often p2p systems have an open clientele, and do not rely on the availability of specific individual machines; rather they can deal with dynamic resources and do not exhibit single points of failure or bottlenecks.

Compared to centralized solutions, the p2p paradigm features a better scalability because the amount of resources grows with the network size, availability (avoiding a single point of failure), reliability, fairness, cooperation incentives, privacy, and security – just about everything researchers expect from a future Internet architecture. As such, it is not surprising that new "clean slate" Internet architecture proposals often revolve around p2p concepts.

One might naively assume that for instance scalability is not an issue in today's Internet, as even most popular web pages are generally highly available. However, this is not necessarily due to our well-designed Internet

David Padua (ed.), *Encyclopedia of Parallel Computing*, DOI 10.1007/978-0-387-09766-4,

architecture, but rather due to the help of so-called overlay networks: The Google Web site for instance manages to respond so reliably and quickly because Google maintains a large distributed infrastructure, essentially a p2p system. Similarly, companies like Akamai sell "p2p functionality" to their customers to make today's user experience possible in the first place. Quite possibly today's p2p applications are just testbeds for tomorrow's Internet architecture.

Implications

p2p networks are often highly dynamic in nature. While traditional computer systems are typically based on fixed infrastructures and are under a single administrative domain (e.g., owned and maintained by a single company or corporation), the participating machines in p2p networks are under the control of individual (and to some extent: anonymous) users who can join and leave at any time and concurrently. In p2p parlor, such membership changes are called *churn*.

A second implication of the autonomy of the machines in p2p networks is that the network consists of different stakeholders. Users can have various reasons for joining the network. For instance, an (anonymous) user may not voluntarily contribute his or her bandwidth, disk space, or CPU cycles to the system, but prefer to *free ride*. This adds a socioeconomic aspect to p2p computing. As the p2p paradigm relies on the contributions of the participating machines, effective incentive mechanisms have to be designed, which foster cooperation and punish free riders.

Another source of inequality in p2p systems apart from selfishness is *heterogeneity*: Due to the open membership, different machines run different operating systems, have different Internet connections, and so on.

Applications

The best-known representatives of p2p technology are probably the numerous file-sharing applications such as Napster, Gnutella, KaZaA, eMule, or BitTorrent. Also, the Internet telephony tool Skype is very popular and used by millions everyday. Zattoo, PPLive, and StreamForge, among many others, use p2p principles to stream video or audio content. The cloud computing service Wuala offers free online storage by exploiting the participants' disks and Internet connections to improve performance. Recently, the power and anonymity of decentralized Internet working has gained the attention of operators of botnets in order to attack certain infrastructure components by a denial-of-service attack. Finally, p2p technology is used for large-scale computer games.

Architecture Variants

Several p2p architectures are known:

- Client/Server goes p2p: Even though Napster is known to the be first p2p system (1999), by today's standards its architecture would not deserve the label p2p anymore. Napster clients accessed a central server that managed all the information of the shared files, i.e., which file was to be found on which client. Only the downloading process itself was between clients ("peers") directly, hence p2p. In the early days of Napster the load of the server was relatively small, so the simple Napster architecture was sufficient. Over time, it turned out that the server may become a bottleneck – and an attractive target for an attack. Indeed, eventually a judge ruled the server to be shut down (a "juridical denial of service attack"). However, it remains to note that many popular P2P networks today still include centralized components, e.g., KaZaA or the eDonkey network accessed by the eMule client. Also, the peer swarms downloading the same file in the BitTorrent network are organized by a so-called tracker whose functionality today is still centralized (although initiatives exist to build distributed trackers).
- Unstructured p2p: The Gnutella protocol is the antithesis of Napster, as it is a fully decentralized system, with no single entity having a global picture. Instead each peer connects to a random sample of other peers, constantly changing the neighbors of this virtual overlay network by exchanging neighbors with neighbors of neighbors. (Any unstructured system also needs to solve the so-called bootstrap *problem*, namely how to discover a first neighbor in a decentralized manner. A popular solution is the use of well-known peer lists.) The fact that users often turn off their clients once they downloaded their content implies high levels of churn (peers joining and leaving at high rates), and hence selecting the right "random" neighbors is an

interesting research problem. The Achilles' heal of unstructured p2p architectures such as Gnutella is the cost of searching. A search request is typically flooded in the network and each search operation will cost *m* messages, *m* being the number of virtual edges in the architecture. In other words, such an unstructured p2p architecture will not scale. Indeed, when Napster was unplugged, Gnutella broke down as well soon afterward due to the inrush of former Napster users.

- Hybrid p2p: The synthesis of client/server architectures such as Napster and unstructured architectures such as Gnutella are hybrid architectures. Some powerful peers are promoted to so-called superpeers (or, similarly, trackers). The set of superpeers may change over time, and taking down a fraction of superpeers will not harm the system. Search requests are handled on the superpeer level, resulting in much less messages than in flat/homogeneous unstructured systems. Essentially, the superpeers together provide a more fault-tolerant version of the Napster server, as all regular peers connect to a superpeer. As of today, almost all popular p2p systems have such a hybrid architecture, carefully trading off reliability and efficiency.
- Structured p2p: Inspired by the early success of Napster, the academic world started to look into the question of efficient file sharing. Indeed, even earlier, in 1997, Plaxton et al. [34] proposed a hypercubic architecture for p2p systems. This was a blueprint for many so-called structured p2p architecture proposals, such as Chord [46], CAN [36], Pastry [37], Tapestry [50], Viceroy [26], Kademlia [27], Koorde [15], SkipGraph [3], and SkipNet [11]. Maybe surprisingly, in practice, structured p2p architectures did not take off yet, apart from certain exceptions such as the Kad architecture (from Kademlia [27]), which is accessible with the eMule client.

Scientific Origins

The scientific foundations of p2p computing were laid many years before the most simple "real" p2p systems like Napster emerged. As already mentioned, in 1997, a blueprint for structured systems has been proposed in [34]. Indeed, also the [34] paper was standing on the shoulders of giants. Some of its eminent precursors are the following:

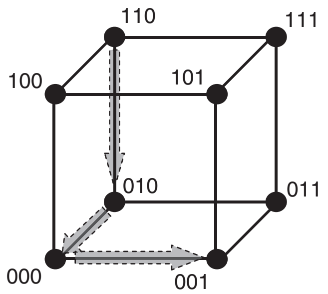
- Research on linear and consistent hashing, e.g., [16].
- Research on locating shared objects, e.g., [4] or [5].
- Research on so-called compact routing: The idea is to construct routing tables such that there is a trade-off between memory (size of routing tables) and stretch (quality of routes), e.g., [31] or [49].
- … and even earlier: hypercubic networks, see below!

Hypercubic Overlays and Consistent Hashing

Every application run on multiple machines needs a mechanism that allows the machines to exchange information. A naive solution is to store at each machine the domain name or IP address of every other machine. While this may work well for a small number of machines, large-scale distributed applications such as file sharing, grid computing, cloud computing, or data center networking systems need a different, more scalable approach: instead of forming a clique (where everybody knows everybody else), each machine should only be required to know some small subset of other machines. This graph of knowledge can be seen as a logical network interconnecting the machines; it is also known as an *overlay network*. A prerequisite for an overlay network to be useful is that it has good topological properties. Among the most important are small peer degree, small network diameter, robustness to churn, or absence of congestion bottlenecks.

The most basic network topologies used in practice are trees, rings, grids, or tori. Many other suggested networks are simply combinations or derivatives of these. The advantage of trees is that the routing is very easy: for every source-destination pair there is only one possible path. However, the root of a tree can be a severe bottleneck. An exception is a p2p streaming system where the single content provider forms the network root. However, trees are also highly vulnerable, e.g., with respect to membership changes.

Essentially all state-of-the-art p2p networks today have some kind of hypercubic topology (e.g., Chord, Pastry, Kademlia). Hypercube graphs have many interesting properties, e.g., they allow for efficient routing: although each peer only needs to store a logarithmic



Peer-to-Peer. Fig. 1 A simplified p2p topology: a three-dimensional hypercube. Each peer has a three-bit identifier. For example, peer 110 is connected to the three peers 010, 100, 111 whose identifiers differ at exactly one position. In order to route a message from peer 110 to say peer 001, one bit is fixed after the other. One possible routing path is depicted in the figure: 110 → 010 → 000 → 001. An alternative path could be 110 → 111 → 101 → 001

number of other peers in the system (the peers' neighbors), by a simple routing scheme, a peer can reach each other peer in a logarithmic number of steps (or "hops"). In a nutshell, this is achieved by assigning each peer a unique d-bit identifier. A peer is connected to all d peers that differ from its identifier at exactly one bit position. In the resulting hypercube network, routing is done by adjusting the bits in which the source and the destination peers differ – one at a time (at most d many). Thus, if the source and the destination differ by k bits, there are $k!$ routes with k hops. Figure 1 gives an example.

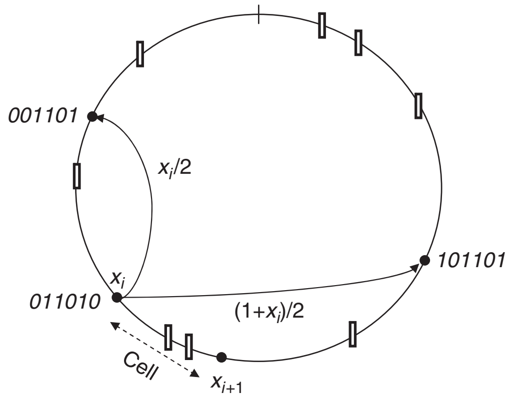
Given a hypercubic topology, it is then simple to construct a distributed hash table (DHT): Assume there are $n = 2^d$ peers that are connected in a hypercube topology as described above. Now a globally known hash function f is used, mapping file names to long bit strings. Let f_d denote the first d bits (prefix) of the bitstring produced by f. If a peer is searching for file name X, it routes a request message $f(X)$ to peer $f_d(X)$. Clearly, peer $f_d(X)$ can only answer this request if all files with hash prefix $f_d(X)$ have been previously registered at peer $f_d(X)$.

There are some additional issues to be addressed in order to design a DHT from a hypercubic topology, in particular how to allow peers to join and leave without notice. To deal with churn the system needs some level of replication, i.e., a number of peers, which are responsible for each prefix such that failure of some peers will not compromise the system. In addition, there are security and efficiency issues that can be addressed to improve the system.

There are many hypercubic networks that are derived from the hypercube: among these are the butterfly, the cube-connected-cycles, the shuffle-exchange, and the de Bruijn graph. For example, the butterfly graph is basically a "rolled out" hypercube (hence directly providing replication!) of constant degree. Another important class of hypercubic topologies are skip graphs [3, 11].

A simple, interesting way to design dynamic p2p systems is the *continuous–discrete approach* described by Naor and Wieder [29]. This approach is based on a "think continuously, act discretely" strategy, and can be used to design a variety of hypercubic topologies. The continuous–discrete approach gives a unified method for performing join/leave operations and for dealing with the scalability issue, thus separating it from the actual network. The idea is as follows: Let I be a Euclidean space, e.g., a (cyclic) one-dimensional space. Let G_c be a graph where the vertex set is the continuous set I. Each point in I is connected to some other points. The actual network then is a discretization of this continuous graph based on a dynamic decomposition of the underlying space I into cells where each "server" is responsible for a cell. Two cells are connected if they contain adjacent points in the continuous graph. Clearly, the partition of the space into cells should be maintained in a distributed manner. When a join operation is performed, an existing cell splits, when a leave operation is performed two cells are merged into one. The task of designing a dynamic and scalable network follows these design rules: (1) Choose a proper continuous graph G_c over the continuous space I. Design the algorithms in the continuous setting, which is often simpler (also in terms of analysis) than in the discrete case. (2) Find an efficient way to discretize the continuous graph in a distributed manner, such that the algorithms designed for the continuous graph would perform well in the discrete graph. The discretization is done via a decomposition of I into the cells. If the cells that compose I are allowed to overlap, then the resulting graph would be fault tolerant.

To give an example, in order to build a dynamic *de Bruijn* network (a so-called Distance Halving DHT), a peer at position $x \in [0,1)$ (in binary form $b_1 b_2$... such



Peer-to-Peer. Fig. 2 The continuous–discrete approach for the dynamic de Bruijn graph. Peers are indicated using circles, files using rectangles. In the continuous setting, the peer at position $x_i = 0.011010$ (in binary notation) is connected to positions $x_i/2$ and $(1 + x_i)/2$. In the discrete setting, it is responsible for the cell (i.e., the connections and files that are mapped there) between positions x_i and x_{i+1}

that $x = \sum_{i=1}^{\infty} 2^{-b_i}$) connects to positions $l(x) := x/2 \in [0,1)$ and $r(x) := (1+x)/2 \in [0,1)$ in G_c (out-degree two per peer). Observe that if position x is written in binary form, then $l(x)$ effectively shifts in a "0" from the left and $r(x)$ shifts in a "1" from the left. Thus, routing is straightforward: based solely on the current position and the destination (without the overhead of maintaining routing tables), a message can be forwarded by a peer by fixing one bit per hop. The set of peers in the cyclic $[0,1)$ space then define the p2p network: Let x_i denote the position of the i^{th} peer (ordered in increasing order with respect to position). Peer i is responsible for the cell $[x_i, x_{i+1})$, computed in a modulo manner, i.e., this peer is responsible to store the data mapped to this cell plus for the establishment of the corresponding connections defined in G_c. Figure 2 gives an example.

Dealing with Churn

A distinguishing property of p2p systems are the frequent membership changes. Measuring the churn levels of existing p2p systems is challenging and one has to be careful when generalizing a given measurement to entire application classes (e.g., [10]). Nevertheless, several insightful measurement studies have been conducted. For instance, [9, 38] reported on the dynamic nature of early p2p networks such as Napster and Gnutella, and [41] analyzed low-level data of a large Internet Service Provider (ISP) to estimate churn. Also the Kad DHT has been subject to measurement studies, and the reader is referred to the results in [47] and [43].

It is widely believed that hypercubic structures are a good basis for churn-resilient p2p systems. As written earlier, a DHT is essentially a hypercubic structure with peers having identifiers such that they span the ID space of the objects to be stored. A simple approach to map the ID space onto the peers has already been described for the hypercube. To give another example, in the butterfly network, we may use its layers for replication, i.e., all peers with the same ID redundantly store the data of the same hash prefix. Other hypercubic DHTs can be more difficult to design, e.g., networks based on the pancake graph [19].

For many well-known systems, theoretic analyses exist showing that the networks remain well-structured after some joins, leaves, or failures occur. In order to evaluate the robustness formally, metrics such as the network *expansion* (for deterministic failures) or the *span* [6] (for randomized failures) are used. Unfortunately, the span is difficult to compute, and the span value is known only for the most simple topologies.

The continuous–discrete approach [29] already mentioned constitutes the basis of several dynamic systems. For example, the SHELL system [40] is robust to certain attacks by connecting older or more reliable peers in a core network where access can be controlled; SHELL also allows to organize heterogeneous peers in an efficient topology.

Many systems proposed in the literature offer a high robustness in the average case, i.e., they provide probabilistic guarantees that hold with high probability. Robustness under attacks or worst-case dynamics is less well understood. In [19], a system is developed that achieves an optimal worst-case robustness in the sense that there is no alternative system that can tolerate higher churn rates without disconnecting. The basic idea is to simulate a hypercube: each peer is part of a distinct hypercube *node*; each hypercube node consists of a logarithmic number of peers. Peers have connections to other peers of their hypercube node and to peers of the neighboring hypercube nodes. After a number of joins and leaves, some peers may have to change to another hypercube node such that up to constant factors, all hypercube nodes have the same cardinality at all times. If the total number of peers grows or shrinks above or below a certain threshold, the dimension of the hypercube is increased or decreased by one, respectively. The balancing of peers among the nodes can be seen as a *dynamic token distribution problem* on the hypercube: Each node of a graph (hypercube) has a certain number of tokens, and the goal is to distribute the tokens along the edges of the graph such that all nodes end up with the same or almost the same number of tokens. Thus, the system builds on two basic components: (1) an algorithm, which performs the described dynamic token distribution and (2) an information aggregation algorithm, which is used to estimate the number of peers in the system and to adapt the hypercube's dimension accordingly. These techniques also work for alternative graphs, like pancake graphs [19].

An appealing notion of robustness is *topological self-stabilization*: A p2p topology is called self-stabilizing if it is guaranteed that from any weakly connected initial state (e.g., after an attack), it will quickly converge to a desirable network in the absence of further membership changes. In contrast to the worst-case churn considered in [19], self-stabilization focuses on the convergence time in periods without membership changes, but allows for general initial system states. While until recently, self-stabilizing algorithms with guaranteed runtime have only been known for simple one-dimensional or two-dimensional linearization problems [14], recently a construction for a variation of skip graphs, namely SKIP+ graphs [13], has been proposed. Single joins and leaves in SKIP+ can be handled locally, and require logarithmic time and polylogarithmic work only. However, there remains the important open question of how to provide degree guarantees during convergence from arbitrary states.

Fostering Cooperation

The appeal of p2p computing arises from the collaboration of the system's constituent parts, the peers. If all the participating peers contribute some of their resources, highly scalable decentralized systems can be built. However, in reality, peers may act selfishly and strive for maximizing their own utility by benefitting from the system without contributing much themselves [42]. Hence, the performance of a p2p system crucially depends on its capability of dealing with selfishness.

Already in 2000, Adar and Huberman [1] noticed that there exists a large fraction of free riders in the file-sharing network Gnutella. The problem of selfish behavior in p2p systems has been a hot topic in p2p research ever since, and many mechanisms to encourage cooperation have been proposed [30]. Perhaps the simplest fairness mechanism is to directly incorporate contribution monitoring into the client software. For instance, in the file-sharing system KaZaA, the client records the contribution of its user. However, such a solution can simply be bypassed by implementing a different client that hard-wires the contribution level to the maximum, as it was the case with *KaZaA Lite*. Inspired by real economies, some researchers have also proposed the introduction of some form of virtual money, which is used for the transactions.

BitTorrent has incorporated a fairness mechanism from the beginning and has hence been subject to intensive research (e.g., [21, 22, 35]). Although this mechanism has similarities to the well-known tit-for-tat scheme, the strategy employed in BitTorrent distinguishes itself from the classic mechanism in many respects. For instance, it is possible for peers to obtain parts of a file "for free," i.e., without reciprocating. While this may be a useful property for bootstrapping newly joined peers, it has been shown that the BitTorrent mechanism can be exploited: the *BitThief* BitTorrent client [24] allows to download entire files fast without uploading any data. It has also been demonstrated in [24] that sharing communities are particularly vulnerable to such exploits. BitThief is not the only client cheating BitTorrent. Piatek et al. [32] presented *BitTyrant*. BitTyrant's strategy is to exploit the BitTorrent protocol in order to maximize download rates. For instance, BitTyrant uses a smart neighbor selection strategy and connects to those peers with the best reciprocation ratios. In contrast to BitThief, BitTyrant does not free ride. BitTyrant seeks to provide the minimal necessary contribution, and also increases the active neighbor set if this is beneficial to the download rate. The authors claim that their client provides a median 70% performance gain in certain environments.

There can be many other forms of strategic behavior in open distributed systems. One subject that has recently gained attention, especially by the game-theoretic research community, is *neighbor selection* in unstructured p2p networks (e.g., [28]). There may be several reasons for a peer to prefer connecting to some peers rather than others. For instance, a peer may want to connect to peers with high bandwidths, peers storing many interesting files, or peers having large degrees and hence provide quick access to many other peers. At the same time, a selfish peer itself may not be eager to store and maintain too many neighbors itself.

Current Trends and Outlook

One can argue that today, p2p computing is already a relatively mature (research) field; nevertheless, there are still many active discussions and developments, also in the context of the future Internet design. Moreover, there exists a discrepancy between the technology of the systems in use and what is actually known in theory. For example, the Kad network is still vulnerable to quite simple attacks [44].

If employed by the wrong people, the flexibility and robustness of p2p technology also constitutes a threat. Denial-of-service attacks are arguably one of the most cumbersome problems in today's Internet, and it is appealing to coordinate botnets in a p2p fashion. A DHT can be used by the bots, e.g., to download new instructions. For instance, it was estimated that in 2007, the DHT-based *Storm botnet* [20] ran on several million computers. Apart from mechanisms to detect or prevent attacks even before they take place, a smart redundancy management may improve availability during the attack itself (see, e.g., the Chameleon system [7]).

In terms of cooperation, there is a tension between the goal of providing incentive compatible mechanisms that exclude free riders and the goal of designing heterogeneous p2p systems that also tolerate (and make use of!) weak participants. Moreover, in addition to design mechanisms dealing with pure selfishness, there is a trend toward p2p systems that are also resilient to malicious behavior (see, e.g., [23] or [39]).

Another active discussion regards the interface between p2p systems and ISPs. The large amount of p2p traffic raises the question of how ISPs should deal with p2p, e.g., by caching contents. p2p networks often employ inefficient overlay-to-ISP mappings as the logical overlay network is typically not aware of the underlying "real" networks and constraints, and much overhead can be avoided by improving the interface between p2p networks and ISPs, e.g., by an oracle [2]. For a critical point of view on the subject, the reader is referred to [33].

It seems that while a few years ago the lion's share of Internet traffic was due to p2p, the proportion seems to be declining [12] now. Especially web services and server-based solutions such as the popular YouTube and RapidShare are catching up. The measured data traces should be interpreted with care however, as they do not take into account what happens behind the scenes of big corporations. Indeed, it is believed that there is a paradigm shift in p2p computing: While p2p retreats (relatively to other applications) from public Internet traffic, today p2p technology plays a crucial role in the coordination and management of large data centers and server farms of corporations such as Akamai or Google.

Related Entries

- ►Cloud Computing
- ►Compact Routing
- ►Consistent Hashing
- ►Decentralization
- ►Distributed Hash Table (DHT)
- ►Grid Computing
- ►Hypercube
- ►Mechanism Design
- ►Open Distributed Systems
- ►Overlay
- ►Overlay Network
- ►Self-organization

Bibliographic Notes and Further Reading

Beyond the specific literature pointed to directly in the text, there are several recommendable introductory books on p2p computing. In particular, the reader is referred to the classic books [8, 45, 48] and two more recent issues [8, 17]. The theoretically more inclined reader may also be interested in [25], which provides an overview of compact routing solutions, and [18] which discusses trade-offs in local algorithms that achieve global goals based on local information only and without centralized entities whatsoever. Regarding the challenges of distributed cooperation, the recent book [30] gives a thorough and up-to-date survey of current (game-theoretic) trends, and also includes a chapter on p2p specific questions.

Bibliography

1. Adar E, Huberman B (2000) Free riding on gnutella. First Monday 5(10):1–22
2. Aggarwal V, Feldmann A, Scheideler C (2007) Can ISPs and p2p users cooperate for improved performance? ACM Comput Commun Rev 37(3):29–40
3. Aspnes J, Shah G (2003) Skip graphs. In: Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms (SODA), Baltimore, 2003
4. Awerbuch B, Peleg D (1990) Sparse partitions. In: Proceedings of the 31st annual symposium on foundations of computer science (SFCS), vol 2, pp 503–513, Washington, 1990
5. Awerbuch B, Peleg D (1995) Online tracking of mobile users. JACM 42(5):1021–1058
6. Bagchi A, Bhargava A, Chaudhary A, Eppstein D, Scheideler C (2004) The effect of faults on network expansion. In: Proceedings of the 16th annual ACM symposium on parallelism in algorithms and architectures (SPAA), Barcelona, 2004
7. Baumgart M, Scheideler C, Schmid S. A DoSresilient information system for dynamic data management. In: Proceedings of the 21st ACM symposium on parallelism in algorithms and architectures (SPAA), Calgary, Alberta, 2009
8. Buford J, Yu H, Lua EK (2008) P2P networking and applications. Morgan Kaufmann, San Francisco, 2008
9. Gummadi K, Dunn R, Saroiu S, Gribble SD, Levy HM, Zahorjan J (2003) Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In: Proceedings of the 19th ACM symposium on operating systems principles (SOSP), Bolton Landing, 2003
10. Haeberlen A, Mislove A, Post A, Druschel P (2006) Fallacies in evaluating decentralized systems. In: Proceedings of the 5th international workshop on peer-to-peer systems (IPTPS), Santa Barbara, 2006
11. Harvey NJA, Jones MB, Saroiu S, Theimer M, Wolman A. Skipnet: a scalable overlay network with practical locality properties. In: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS), Seattle, 2003
12. IPOQUE (2009) Internet study 2008/2009. http://www.ipoque.com/resources/internet-studies/internet-study-2008-2009, pp 704–713 (accessed on October 31, 2010)
13. Jacob R, Richa A, Scheideler C, Schmid S, Täubig H (2009) A distributed polylogarithmic time algorithm for self-stabilizing skip graphs. In: Proceedings of the ACM symposium on principles of cistributed computing (PODC), New York, 2009
14. Jacob R, Ritscher S, Scheideler C, Schmid S (2009) A self-stabilizing and local delaunay graph construction. In: Proceedings of the 20th international symposium on algorithms and computation (ISAAC), Hawaii, 2009
15. Kaashoek F, Karger DR (2003) Koorde: a simple degree-optimal distributed hash table. In: Proceedings of the international workshop on peer-to-peer systems (IPTPS), Berkeley, 2003
16. Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the 29th ACM symposium on theory of computing (STOC), New York, pp 654–663, 1997
17. Khan J, Wierzbicki A (2008) Foundation of peer-to-peer computing. Elsevier Computer Communication, 2008
18. Kuhn F, Moscibroda T, Wattenhofer R (2006) The price of being near-sighted. In: Proceedings of the 17th ACM-SIAM symposium on discrete algorithms (SODA), Miami, 2006
19. Kuhn F, Schmid S, Wattenhofer R (2010) Towards worstcase churn resistant peer-to-peer systems. J Distrib Comput (DIST) 22(4):249–267
20. Larkin E (2007) Storm worm's virulence may change tactics. British Computer Society (accessed on August 03, 2007)
21. Legout A, Urvoy-Keller G, Michiardi P (2006) Rarest first and choke algorithms are enough. In: Proceedings of the 6th ACM SIGCOMM conference on internet measurement (IMC), pp 203–216, Rio de Janeriro, 2006
22. Levin D, LaCurts K, Spring N, Bhattacharjee B (2008) Bittorrent is an auction: analyzing and improving bittorrent's incentives. SIGCOMM Comput Commun Rev 38(4):243–254
23. Li H, Clement A, Marchetti M, Kapritsos M, Robinson L, Alvisi L, Dahlin M (2008) Flightpath: obedience vs choice in cooperative services. In: Proceedings of the symposium on operating systems design and implementation (OSDI), San Diego, 2008
24. Locher T, Moor P, Schmid S, Wattenhofer R (2006) Free riding in bittorrent is cheap. In: Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets), Irvine, 2006
25. Malkhi D (2004) Locality-aware network solutions. Technical Report, The Hebrew University of Jerusalem, HUJI-CSE-LTR-2004-6
26. Malkhi D, Naor M, Ratajczak D (2002) Viceroy: a scalable and dynamic emulation of the buttery. In: Proceedings of the 21st annual symposium on principles of distributed computing (PODC), Monterey, 2002
27. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the xor metric. In: Proceedings of the 1st international workshop on peer-to-peer systems (IPTPS), Cambridge, 2002
28. Moscibroda T, Schmid S, Wattenhofer R (2006) On the topologies formed by selfish peers. In: Proceedings of the 25th annual symposium on principles of distributed computing (PODC), Denver, 2006
29. Naor M, Wieder U (2003) Novel architectures for p2p applications: the continuous-discrete approach. In: Proceedings of the 15th annual ACM symposium on parallel algorithms and architectures (SPAA), pp 50–59, San Diego, 2003
30. Nisan N, Roughgarden T, Tardos E, Vazirani VV (2007) Algorithmic game theory. Cambridge University Press, Cambridge
31. Peleg D, Upfal E (1988) A tradeoff between space and efficiency for routing tables. In: Proceedings of the 20th annual ACM symposium on theory of computing (STOC), pp 43–52, Chicago, 1988

32. Piatek M, Isdal T, Anderson T, Krishnamurthy A, Venkataramani A (2007) Do incentives build robustness in bittorrent? In: Proceedings of the 4th USENIX symposium on networked systems design and implementation, Cambridge, 2007
33. Piatek M, Madhyastha HV, John JP, Krishnamurthy A, Anderson T (2009) Pitfalls for ISP-friendly p2p design. In: Proceedings of the hotnets, New York, 2009
34. Plaxton C, Rajaraman R, Richa AW (1997) Accessing nearby copies of replicated objects in a distributed environment. In: Proceedings of the 9th ACM symposium on parallel algorithms and architectures (SPAA), pp 311, 320, Newport, 1997
35. Qiu D, Srikant R (2004) Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: Proceedings of the conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM), pp 367–378, New York, 2004
36. Ratnasamy S, Francis P, Handley M, Karp R, Schenker S (2001) A scalable content-addressable network. In: Proceedings of the ACM SIG-COMM conference on applications, technologies, architectures, and protocols for computer communications, pp 161–172, New York, 2001
37. Rowstron AIT, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM international conference on distributed systems platforms (middleware), pp 329–350, Heibelberg, 2001
38. Saroiu S, Gummadi PK, Gribble SD (2002) A measurement study of peer-to-peer file sharing systems. In: Proceedings of the multimedia computing and networking (MMCN), San Jose, 2002
39. Scheideler C (2005) How to spread adversarial nodes?: rotate! In: Proceedings of the 37th Annual ACM symposium on theory of computing (STOC), pp 704–713, Baltimore, 2005
40. Scheideler C, Schmid S (2009) A distributed and oblivious heap. In: Proceedings of the 36th international colloquium on automata, languages and programming (ICALP), Rhodes, 2009
41. Sen S, Wang J (2004) Analyzing peer-to-peer traffic across large networks. IEEE/ACM Trans Netw 12(2):219–232
42. Shneidman J, Parkes DC (2003) Rationality and self-interest in peer to peer networks. In: Proceedings of the 2nd international workshop on peer-to-peer systems (IPTPS), Berkeley, 2003
43. Steiner M, Biersack EW, Ennajjary T (2007) Actively monitoring peers in KAD. In: Proceedings of the 6th international workshop on peer-to-peer systems (IPTPS), Bellevue, 2007
44. Steiner M, En-Najjary T, Biersack EW (2007) Exploiting KAD: possible uses and misuses. Comput Commu Rev 37(5):65–69
45. Steinmetz R, Wehrle K (2005) Peer-to-peer systems and applications. Springer, Heidelberg
46. Stoica I, Morris R, Karger D, Kaashoek F, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SIGCOMM conference on applications, technologies, architectures, and protocols for computer communications, San Diego, 2001
47. Stutzbach D, Rejaie R (2006) Understanding churn in peer-to-peer networks. In: Proceedings of the 6th internet measurement conference (IMC), New York, 2006
48. Subramanian R, Goodman B (2005) Peer-to-peer computing: the evolution of a disruptive technology. IGI, Hershey
49. Thorup M, Zwick U (2001) Compact routing schemes. In: Proceedings of the annual ACM symposium on parallel algorithms and architectures (SPAA), pp 1–10, Crete, Greece, 2001
50. Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD, Kubiatowicz J (2004) Tapestry: a resilient global-scale overlay for Service deployment. IEEE journal on selected areas in commonuincations Vol. 22, No. 1