Chapter 14

Peer-to-Peer Computing

“Indeed, I believe that virtually every important aspect of programming arises somewhere in the context of sorting and searching!”

– Donald E. Knuth, The Art of Computer Programming

14.1 Introduction

Unfortunately, the term peer-to-peer (P2P) is ambiguous, 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, often of dubious origin. In the early days, P2P users mostly shared music, pictures, and software; nowadays books, movies or tv shows have caught on. P2P file sharing is immensely popular, currently at least half 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 (file sharing) content 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 grey area, e.g., P2P Internet telephony à la Skype, P2P mass player games on video consoles connected to the Internet, P2P live video streaming as in Zattoo or StreamForge, or P2P social storage such as Wuala. So, again, what is P2P?! Still not an easy question... 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 we learn in this course is P2P! Moreover, according to this definition early-day file sharing applications such as Napster (1999) that essentially made the term P2P popular would not be P2P! On the other hand, the plain old telephone system or the world wide web do fit the P2P definition...

• 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 unencumbered. (Studies show that these freedom-of-speech P2P networks do not feature a lot of content against oppressive governments; indeed the majority of text documents seem to be about illicit drugs, not to speak about the type of content in audio or video files.)

In other words, we cannot hope for a single well-fitting definition of P2P, as some of them even contradict. In the following we mostly employ the academic viewpoints (second and third definition above). In this context, it is generally believed that P2P will have an influence on the future of the Internet. The P2P paradigm promises to give better scalability, availability, reliability, fairness, 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 really because of our well-designed Internet architecture, but rather due to the help of so-called overlay networks. The Google website 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.

14.2 Architecture Variants

Several P2P architectures are known:

• Client/Server goes P2P: Even though Napster is known to be the 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 peer-to-peer. In the early days of Napster the load of the server was relatively small, so the simple Napster architecture made a lot of sense. Later on, it became clear that the server would eventually be a bottleneck, and more so as an attractive target for an attack. Indeed, eventually a judge ruled the server to be shut down, in other words, he conducted a juridical denial of service attack.

• 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 would connect to a random sample of other
peers, constantly changing the neighbors of this virtual overlay network by exchanging neighbors with neighbors of neighbors. In such a system it is part of the challenge to find a decentralized way to even discover a first neighbor; this is known as the bootstrap problem. To solve it, usually some random peers of a list of well-known peers are contacted first. When searching for a file, the request was being flooded in the network (Algorithm 11 in Chapter 3). Indeed, since users often turn off their client once they downloaded their content there usually is a lot of churn (peers joining and leaving at high rates) in a P2P system, so selecting the right \( \text{random} \) neighbors is an interesting research problem by itself. However, unstructured P2P architectures such as Gnutella have a major disadvantage, namely that each search 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.

- 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, 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, but essentially not using any fancy algorithms and techniques.

- Structured P2P: Inspired by the early success of Napster, the academic world started to look into the question of efficient file sharing. The proposal of hypercubic architectures lead to many so-called structured P2P architecture proposals, such as Chord, CAN, Pastry, Tapestry, Viceroy, Kademia, Koorde, SkipGraph, SkipNet, etc. In practice structured P2P architectures are not yet popular, apart from the Kad (from Kademia) architecture which comes for free with the eMule client.

### 14.3 Hypercubic Networks

In this section we will introduce some popular families of network topologies. These topologies are used in countless application domains, e.g., in classic parallel computers or telecommunication networks, or more recently (as said above) in P2P computing. Similarly to Chapter 4 we employ an All-to-All communication model, i.e., each node can set up direct communication links to arbitrary other nodes. Such a virtual network is called an overlay network, or in this context, P2P architecture. In this section we present a few overlay topologies of general interest.

The most basic network topologies used in practice are trees, rings, grids or tori. Many other suggested networks are simple 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 simple path. However, since the root of a tree is usually a severe bottleneck, so-called fat trees have been used. These trees have the property that every edge connecting a node \( u \) to its parent \( v \) has a capacity that is equal to all leaves of the subtree rooted at \( v \). See Figure 14.1 for an example.

![Figure 14.1: The structure of a fat tree.](image)

**Remarks:**

- Fat trees belong to a family of networks that require edges of non-uniform capacity to be efficient. Easier to build are networks with edges of uniform capacity. This is usually the case for grids and tori. Unless explicitly mentioned, we will treat all edges in the following to be of capacity 1.
- In the following, \( [x] \) means the set \( \{0, \ldots, x - 1\} \).

**Definition 14.1 (Torus, Mesh).** Let \( m, d \in \mathbb{N} \). The \( (m, d) \)-torus \( T(m, d) \) is a graph with node set \( V = [m]^d \) and edge set

\[
E = \left\{ \{(a_1, \ldots, a_d), (b_1, \ldots, b_d)\} \mid a_i, b_i \in [m], \sum_{i=1}^{d} |a_i - b_i| = 1 \right\}
\]

The \( (m, d) \)-torus \( T(m, d) \) is a graph that consists of an \( (m, d) \)-mesh and additionally wrap-around edges from nodes \( (a_1, \ldots, a_{d-1}, m, a_{d+1}, \ldots, a_d) \) to nodes \( (a_1, \ldots, a_{d-1}, 1, a_{d+1}, \ldots, a_d) \) for all \( i \in \{1, \ldots, d\} \) and all \( a_j \in [m] \) with \( j \neq i \).

In other words, we take the expression \( a_i - b_i \) in the sum modulo \( m \) to compute the absolute value. \( M(m, 1) \) is also called a line, \( T(1, 1) \) a cycle, and \( M(2, 2) = T(2, 2) \) a \( d \)-dimensional hypercube.

**Figure 14.2 presents a linear array, a torus, and a hypercube.**

**Remarks:**

- Routing on mesh, torus, and hypercube is trivial. On a \( d \)-dimensional hypercube, to get from a source bitstring to a target bitstring one only needs to fix each “wrong” bit, one at a time; in other words, if the source and the target differ by \( k \) bits, there are \( k! \) routes with \( k \) hops.
Remarks:

• Figure 14.3 shows the 3-dimensional butterfly BF(3). The BF(d) has 
  \( \binom{d+1}{2} \cdot 2^d \) nodes and degree 4. It is not difficult to check that 
  \( BF(d) \) has the same structure as \( W-BF(d) \), with the hypercube \( W \) 
  identified as a single node, i.e., \( BF(d) \) is a rolled-out hypercube. In 
  this construction, the butterfly network is nothing but a folded-back 
  hypercube, the nodes of the butterfly are exactly the nodes of the 
  corresponding hypercube, and the edges of the butterfly network are 
  obtained from the hypercube by adding a new edge between any two 
  nodes \( v \) and \( w \) of the hypercube such that \( \text{dist}([v], [w]) = 1 \). This 
  construction is often called the butterfly network. 

• The structure of a butterfly network is easily seen from the 
  butterfly image. They have been used decades earlier for communication 
  switches. 

• Butterflies have the advantage of a constant node degree over 
  hypercubes, whereas hypercubes feature more fault-tolerant routing.

• The hypercube has many derivatives, the so-called hypercubic 
  networks. Among these are the butterfly, cube-connected-cycles, 
  shuffle-exchange, and de Bruijn graph. We start with the butterfly, which 
  is basically a "rolled out" hypercube (hence directly providing replication!).

14.3 HYPERCUBIC NETWORKS

Definition 14.2 (Butterfly). Let \( d \in \mathbb{N} \). The \( d \)-dimensional butterfly 
  \( BF(d) \) is a graph with node set \( V = \{0, 1, \ldots, 2^d - 1\} \) and edge set 
  \( E = \{\{(i, \alpha), (i+1, \beta)\} : i \in \{0, 1, \ldots, 2^d - 1\}, \alpha, \beta \in \{0, 1\}, \alpha \neq \beta\} \). 

Definition 14.3 (Cube-Connected-Cycles). Let \( d \in \mathbb{N} \). The 
  cube-connected-cycles network CCC(d) is a graph with node set 
  \( V = \{(a, p) : a \in \{0, 1\}^d, p \in \{0, 1, \ldots, d - 1\}\} \) and edge set 
  \( E = \{(i, \alpha), (i+1, \beta) : i \in \{0, 1, \ldots, 2^d - 1\}, \alpha, \beta \in \{0, 1\}, \alpha \neq \beta\} \).
14.3. HYPERCUBIC NETWORKS

CHAPTER 14. PEER-TO-PEER COMPUTING

Figure 14.6: The structure of \(DB(2,2)\) and \(DB(2,3)\).

Remarks:
- Two examples of a DeBruijn graph can be found in Figure 14.6. The DeBruijn graph is the basis of the Koorde P2P architecture.
- There are some data structures which also qualify as hypercubic networks. An obvious example is the Chord P2P architecture, which uses a slightly different hypercubic topology. A less obvious (and therefore good) example is the skip list, the balanced binary search tree for the lazy programmer.

Definition 14.4 (Shuffle-Exchange). The shuffle-exchange graph \(SE(d)\) is defined as an undirected graph with node set \(V = [2]^d\) and an edge set \(E = E_1 \cup E_2\) with

\[
E_1 = \{ \{(a_1, \ldots, a_d), (a_1, \ldots, a_d)\} \mid (a_1, \ldots, a_d) \in [2]^d, a_d = 1 - a_d \}
\]

and

\[
E_2 = \{ \{(a_1, \ldots, a_d), (a_1, a_2, \ldots, a_{d-1})\} \mid (a_1, \ldots, a_d) \in [2]^d \}.
\]

Figure 14.5 shows the 3- and 4-dimensional shuffle-exchange graph.

Figure 14.4: The structure of CCC(3).

Remarks:
- Two possible representations of a CCC can be found in Figure 14.4.
- The shuffle-exchange is yet another way of transforming the hypercubic interconnection structure into a constant degree network.

Definition 14.5 (DeBruijn). The \(b\)-ary DeBruijn graph of dimension \(d\) \(DB(b,d)\) is an undirected graph \(G = (V,E)\) with node set \(V = \{v \in [b]^d\}\) and edge set \(E\) that contains all edges \(\{(v, w) : v \in [b]^d\}\) with the property that \(w \in [(v_1, \ldots, v_{d-1}) : x \in [b]]\), where \(v = (v_1, \ldots, v_d)\).

Figure 14.5: The structure of \(SE(3)\) and \(SE(4)\).

Remarks:
- Search, insert, and delete can be implemented in \(O(\log n)\) expected time in a skip list, simply by jumping from higher levels to lower ones when overshooting the searched position. Also, the amortized memory cost of each object is constant, as on average an object only has two forward pointers.
- The randomization can easily be discarded, by deterministically promoting a constant fraction of objects of level \(i\) to level \(i+1\), for all \(i\). When inserting or deleting, object \(o\) simply checks whether its left and right level \(i\) neighbors are being promoted to level \(i+1\). If none of them is, promote object \(o\) itself. Essentially we establish a MIS on each level, hence at least every third and at most every second object is promoted.
- There are obvious variants of the skip list, e.g., the skip graph. Instead of promoting only half of the nodes to the next level, we always promote all the nodes, similarly to a balanced binary tree: All nodes are part of the root level of the binary tree. Half the nodes are promoted left, and half the nodes are promoted right, on each level. Hence on level \(i\) we have \(2^i\) lists (or, more symmetrically: rings) of about \(n/2^i\) objects. This is pretty much what we need for a nice hypercubic P2P architecture.
- One important goal in choosing a topology for a network is that it has a small diameter. The following theorem presents a lower bound for this.

\[
DB = \{ (a_1, \ldots, a_d) : (a_1, \ldots, a_d) \in [2]^d, a_d = 1 - a_d \}
\]

and

\[
DB = \{ (a_1, \ldots, a_d), (a_1, a_2, \ldots, a_{d-1}) : (a_1, \ldots, a_d) \in [2]^d \}.
\]
Theorem 14.7. Every graph of maximum degree \( d > 2 \) and size \( n \) must have a diameter of at least \( \left\lceil \frac{\log n}{\log(d-1)} \right\rceil - 2 \).

Proof. Suppose we have a graph \( G = (V,E) \) of maximum degree \( d \) and size \( n \). Start from any node \( v \in V \). In a first step at most \( d \) other nodes can be reached. In two steps at most \( d(d-1) \) additional nodes can be reached. Thus, in general, in at most \( k \) steps, at most

\[
1 + \sum_{i=0}^{k-1} d(i^{-1}) = 1 + d \sum_{i=0}^{k-1} \frac{d^{-1}}{i+1} \leq \frac{d}{d-1} \cdot \frac{d^k - 1}{d-2}
\]

can be reached. This has to be at least \( n \) to ensure that \( v \) can reach all other nodes in \( V \) within \( k \) steps. Hence,

\[
\frac{(d-1)^k}{d} \geq \frac{(d-2) n}{d} \iff k \geq \log_{d-1} (\frac{(d-2)n}{d}).
\]

Since \( \log_{d-1} (\frac{(d-2)n}{d}) > -2 \) for all \( d > 2 \), this is true only if \( k \geq \left\lceil \frac{\log n}{\log(d-1)} \right\rceil - 2 \).

Remarks:

- In other words, constant-degree hypercubic networks feature an asymptotically optimal diameter.

- There are a few other interesting graph classes, e.g., expander graphs (an expander graph is a sparse graph which has high connectivity properties, that is, from every not too large subset of nodes you are connected to a larger set of nodes), or small-world graphs (popular representations of social networks). At first sight hypercubic networks seem to be related to expanders and small-world graphs, but they are not.

14.4 DHT & Churn

As written earlier, a DHT essentially is a hypercubic structure with nodes having identifiers such that they span the ID space of the objects to be stored. We described the straightforward way how the ID space is mapped onto the peers for the hypercube. Other hypercubic structures may be more complicated. The butterfly network, for instance, may directly use the \( d+1 \) layers for replication, i.e., all the \( d+1 \) nodes with the same ID are responsible for the same hash prefix. For other hypercubic networks, e.g., the pancake graph (see exercises), assigning the object space to peer nodes may be more difficult.

In general a DHT has to withstand churn. Usually, peers are under control of individual users who turn their machines on or off at any time. Such peers join and leave the P2P system at high rates ("churn"), a problem that is not existent in orthodox distributed systems, hence P2P systems fundamentally differ from old-school distributed systems where it is assumed that the nodes in the system are relatively stable. In traditional distributed systems a single unavailable node is a minor disaster: all the other nodes have to get a consistent view of the system again, essentially they have to reach consensus which nodes are available.
The advantage of prefix-based over more rigid DHT structures is that there is a large choice of neighbors for most prefixes. Peers are no longer bound to connect to peers exactly matching a given identifier. Instead, peers are enabled to connect to any peer matching a desired prefix, regardless of subsequent identifier bits. In particular, among half of all peers can be chosen for a shared prefix of length 0. The flexibility of such a neighbor policy allows the optimization of prefix further down in its distribution branch. Such overlay structures are not limited to distributed storage. Instead, they are equally well suited for the distribution of content, such as multicasting of radio stations or television channels. In a basic multicasting scheme, a source with identifier 0000 may forward new data blocks to two peers having identifiers starting with 0 and 1. They in turn forward the content to peers having identifiers starting with 00, 01, 10, and 11. The recursion finishes once all peers are reached. This basic scheme has the subtle shortcoming that data blocks may pass by multiple times at a single peer because a predecessor can match a prefix further down in its distribution branch. The subsequent multicasting scheme avoids this problem by modifying the topology and using a different routing scheme. For simplicity, the neighbor selection policy is presented for the case \( \beta = 1 \). In order to use \( M \), the peers must store links to a different set of neighbors. A peer \( v \) with the identifier \( b_0 \ldots b_{\beta} \ldots b_{d-1} \) stores links to peers whose identifiers start with \( b_0 \ldots b_{\beta} \ldots b_i \) for all \( \ell_i \). For example, the peer with the identifier 0000 has to maintain connections to peers whose identifiers start with the prefixes 00, 01, 010, 011, 0010, and 0011. Pseudo-code for the algorithm is given in Algorithm 56. The parameters are the length \( n \) of the prefix that is not to be modified and at most one critical predecessor \( v_c \). If \( \beta = 1 \), any node \( v \) tries to forward the data block to two peers \( v_1 \) and \( v_2 \). The procedure is called at the source \( v \) with arguments \( v = v_1 = v_2 = 0 \), resulting in the two messages \( \text{forward}(1,0) \) to \( v_1 \) and \( \text{forward}(1,0) \) to \( v_2 \). The peer \( v_1 \) is chosen locally such that the prefix its identifier shares with the identifier of \( v \) is the shortest among all those whose shared prefix length is at least \( \ell \). This value \( f(v_1, v) \) and \( v \) itself are the parameters included in the forward message to peer \( v_1 \), if such a peer exists. The second peer is chosen similarly, but with respect to \( v_2 \) and not \( v \). If no suitable peer is found in the routing table, the peer \( v_c \) is queried for a candidate using the subroutine get\text{Next} which is described in Algorithm 57. This step is required because node \( v \) cannot deduce from its routing table whether a peer \( v_2 \) with the property \( f(v_2, v) \geq \ell \) exists. In the special case when \( v_1 = 0 \), \( v_2 \) is chosen locally, if possible, such that \( f(v_2, v) = \pi \). In Figure 14.8, a sample
14.5. STORAGE AND MULTICAST

Algorithm 56 \( M \): forward(\( \pi, v \)) at peer \( v \).

\[1: S := \{v' \in N_v : f(v', v) \geq \pi + 1\} \]
\[2: \text{choose } v_1 \in S; f(v_1, v) \leq f(v', v) \forall v' \in S \]
\[3: \text{if } v_1 \neq 0 \text{ then} \]
\[4: \text{forward}(f(v_1, v), v) \to v_1 \]
\[5: \text{end if} \]
\[6: \text{if } v_2 \neq 0 \text{ then} \]
\[7: \text{choose } v_2 \in N_v; f(v_2, v) = \pi + 1 \]
\[8: \text{if } v_2 = 0 \text{ then} \]
\[9: v_2 := \text{getNext}(v) \text{ from } v_2 \]
\[10: \text{end if} \]
\[11: \text{if } v_2 \neq 0 \text{ then} \]
\[12: \text{forward}(f(v_2, v), v) \to v_2 \]
\[13: \text{end if} \]
\[14: \text{else} \]
\[15: \text{choose } v_2 \in N_v; f(v_2, v) = \pi \]
\[16: \text{if } v_2 \neq 0 \text{ then} \]
\[17: \text{forward}(\pi + 1, v) \to v_2 \]
\[18: \text{end if} \]
\[19: \text{end if} \]

The presented multicasting scheme \( M \) has the property that, at least in a static setting, wherein peers neither join nor leave the overlay, all peers can be reached and each peer receives a data block exactly once as summarized by the following theorem:

**Theorem 14.9.** In a static overlay, algorithm \( M \) has the following properties:

(a) It does not induce any duplicate messages (loop-free), and

(b) all peers are reached (complete).

Remarks:

- The multicast scheme \( M \) benefits from the same overlay properties as DHTs; there is a bounded diameter and peer degree. Peers can maintain backup neighbors and favor low-latency, high-bandwidth peers as neighbors. Most importantly, intermediate peers have the possibility to choose among multiple (backup) neighbors to forward incoming data blocks. This, in turn, allows peers to quickly adapt to changing network conditions such as churn and congestion. It is not necessary to rebuild the overlay structure after failures. In doing so, a system can gain both robustness and efficiency.

- In contrast, for more rigid data structures, such as trees, data blocks are forced to travel along fixed data paths, rendering them susceptible to any kind of failure.

- Conversely, unstructured and more random overlay networks lack the structure to immediately forward incoming data blocks. Instead, such systems have to rely on the exchange of periodic notifications about available data blocks and requests and responses for the download of missing blocks, significantly increasing distribution delays. Furthermore, the lack of structure makes it hard to maintain connectivity among all peers. If the neighbor selection is not truly random, but based on other criteria such as latency and bandwidth, clusters may form that disconnect themselves from the remaining overlay.

There is a variety of further flavors and optimizations for prefix-based overlay structures. For example, peers have a logarithmic number of neighbors in the presented structure. For 100,000 and more peers, peers have at least 20 neighbors. Selecting a backup neighbor doubles the number of neighbors to 40. Using \( M \) further doubles their number to 80. A large number of neighbors accrues substantial maintenance costs. The subsequent variation limits the number of neighbors with a slight adjustment of the overlay structure. It organizes peers into disjoint groups \( G_0, G_1, \ldots, G_m \) of about equal size. The introduction of groups is motivated by the fact that they will enable peers to have neighboring connections for a subset of all shared prefixes while maintaining the favorable overlay properties. The source, feeding blocks into the overlay, joins group \( G_0 \). The other peers randomly join groups. Let \( g(v) \) denote the function that assigns each peer \( v \) to a group, i.e., \( v \in G_{g(v)} \).

Peers select neighboring peers based not solely on shared prefixes but also on group membership. A peer \( v \) with the identifiers \( b_0 \), \( b_1 \) stores links to neighboring peers whose identifiers start with \( b_0^k \) \( b_1^k \) and belong to group \( g(v) + 1 \mod m \) for all \( i \in (g(v), g(v) + m, g(v) + 2m, g(v) + 3m, \ldots) \). Furthermore, let \( f \) denote the first index \( i \) where no such peer exists. As fallback, peer \( v \) stores further links to peers from arbitrary groups whose identifiers start with \( b_0^k b_1^k \) for all \( k \geq f - m + 1 \). The fallback connections allow a peer to revert to the regular overlay structure for the longest shared prefixes where only few peers exist.
As an example, a scenario with \( m = 4 \) groups is considered. A peer with identifier 00...0 belonging to group \( G \) has to maintain connections to peers from group \( G_2 \) that share the prefixes 001, 0000001, 00000000001, etc. In an overlay with 100 peers, the peer is unlikely to find a neighbor for a prefix length larger than \( \log(100) \), such as prefix 00000000001. Instead, he further maintains fallback connections to peers from arbitrary groups having identifiers starting with the prefixes 000000000001, 000000000001, etc. (if such peers exist).

**Remarks:**
- By applying the presented grouping mechanism, the total number of neighbors is reduced to \( \frac{\log n}{m} + c \) with constant \( c \) for fallback connections. (Note that peers have both outgoing neighbors to the next group and incoming neighbors from the previous group, doubling the number of neighbors.)
- Setting the number of groups \( m = \log n \) gives a constant number of neighbors regardless of the overlay size.

**Chapter Notes**

The paper of Plaxton, Rajaraman, and Richa [PRR97] laid out a blueprint for many so-called structured P2P architecture proposals, such as Chord [SMK94], CAN [RFH97], Pastry [RD01], Vicent [MM02], Roode [KK93], SkipGraph [AS03], SkipNet [HJS03], or Tapestry [ZHS04]. Also the paper of Plaxton et. al. was standing on the shoulders of giants. Some of its eminent precursors are: linear and consistent hashing [KLL+97], locating shared objects [AP90, AP91], compact routing [SK85, PU88], and even earlier: hypercubic networks, e.g. [AJ75, Wirt81, GS81, BAK+].

Furthermore, the techniques in use for prefix-based overlay structures are related to a proposal called LAND, a locality-aware distributed hash table proposed by Abraham et al. [AMD04].

More recently, a lot of P2P research focused on security aspects, describing instance attacks [MSW06, SAE07, LAC07], and possible countermeasures [KSW05, AS09, BSW09]. There are several recommendable introductory books on P2P computing, e.g. [SW06, SG05, MS07, KW08, BYL08].

Some of the figures in this chapter have been provided by Christian Scheideler.

**Bibliography**


[LMSW06] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free Riding in BitTorrent is Cheap. In 5th Workshop on Hot Topics in Networks (HotNets), Irvine, California, USA, November 2006.


