Peer-to-Peer Overlay Networks: A Survey

Chonggang Wang Bo Li

Department of Computer Science
The Hong Kong University of Science and Technology, Hong Kong
{cgwang, bli}@cs.ust.hk

April 20, 2003

Abstract-P2P network is factually an overlay network for distributed object store, search and sharing. This paper will present a survey about it's recent. First, we will introduce a simple definition of P2P, and define some common operation procedures in P2P. Second, P2P performance metric will be discussed in order to understand and differentiated the practical P2P architectures and protocols. Third, the current P2P architecture will be classified and compared in detail. Then, object search protocols will be carefully discussed with corresponding P2P architecture. We also review the current P2P modeling outputs and list the general modeling approaches. Finally, several novel applications based on P2P techniques and future research directions of P2P overlay networks are briefly investigated. This paper aims to give a clear and complete survey of P2P networks according to clues of its development and evolution process.

1 Introduction

Oram [1] gives a simple definition of peer-to-peer (P2P) as: "P2P is a class of applications that take advantage of resources storage, cycles, content, human presence available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, peer-to-peer nodes must operate outside the DNS and have significant or total autonomy of central servers". In a short word, P2P is a special distributed system on the application layer, where each pair of peers can communicate each other through the routing protocol in P2P layers. A general P2P network is pictured in figure 1. Each node (peer) keeps an object (such as file, music, MP3, MPEG, etc.) database. Each peer can query its wanted object from other peers through the logical connection on P2P layer.

In order to understand P2P clearly, we can compare P2P model with C/S model (Table 1). In C/S model, each node takes role of either client or server. The resource such as storage or computing capability can be shared between client and server. The server in C/S model is factually a central control point. In P2P model, each node takes role of client and server. As a client, it can query and download its wanted objects from other nodes (peers). As a server, it can provide objects to other nodes at the same time.

According to lifetime cycle, there are general four phases for a P2P node: join, query, download, and depart. First, a just arriving node must actively join the P2P system. In this procedure, it may get some basic information (such as its neighbors) to start up, and simultaneously publish information about the objects it holds. Then the node can issue query for objects it wants. At this time, P2P location protocol will help the node to determine destination node, while P2P routing protocol can route query messages to destination node. Third, the node can directly download the object from the destination node if the query successes (often returns the IP address of destination node). Finally, the node will announce its departure in the ideal case. So three important components of P2P are: *neighbor finding, location protocol, and routing protocol*. These components will use the P2P topology, which is structured or unstructured, and self-organized by P2P nodes themselves and will be no use when beginning to download objects.

This paper aims to give out a survey of P2P recent advances. Although there are some overview papers [2-4] about P2P networks, we emphasize on P2P architecture, object query (search) algorithms, and P2P performance evaluating and modeling. We try to give out a complete and clear introduction and analysis about today's P2P network, their performance comparison, unsolved problems, and some future directions. The remainder of this paper is organized as follows. Several P2P performance metrics will be listed in section 2. In section 3, we will discuss and compare the

* The work was supported by in part by grants from Hong Kong Research Grants Council (RGC) under contracts AoE/E-01/99 and HKUST 6192/02E.

current P2P architectures in three generations. Then object search algorithm, one of the most important components in P2P networks, is carefully described in section 4. Section 5 will discuss some progresses about P2P performance modeling. Finally, some P2P key applications and future directions are summarized respectively in section 6 and section 7. Some conclusions are presented in section 8.

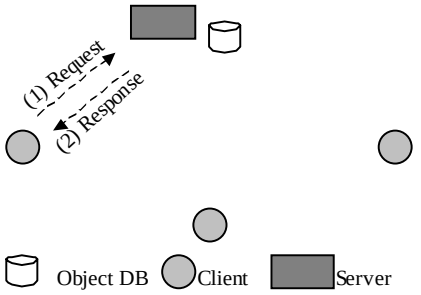


Figure 1 Traditional C/S model

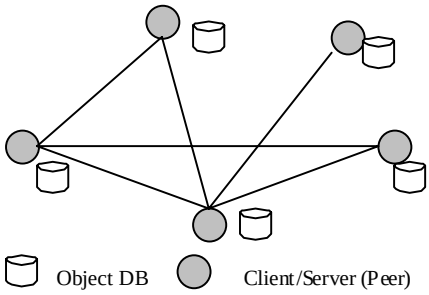


Figure 2 General P2P model

Model	Role of node	Sharing Relationship	Central Control
C/S	Client or Server	C<->S	Yes (Server)
P2P	Servent*	Peer<->Peer	No

Table 1 Comparison of P2P and C/S model (*Client and Server)

2 P2P Performance Metrics

In order to evaluate and compare the proposed P2P networks, some performance metric must be defined first. The common properties of a P2P networks usually include: *security*, *anonymity*, *scalability*, *resilience*, and *query efficiency*. This paper will pay attentions to last three aspects. The *resilience* can be measured by *query-hit ratio*. We can call nodes issuing query as requesters, and nodes holding the requested objects as providers (figure 3). There are two cases that will bring failures: 1) Providers are down. 2) There are failure on the path to providers (such as link failure or re-lay node failure). However, *Query efficiency* is dependent on *average query message number* and *average query path length*. Generally speaking, object query algorithm influences query message number. If using flooding-based forwarding, query message number will be certainly bigger than that using single-route forwarding. But flooding-based forwarding will bring with small query path length. Some other metrics may comprise: 1) the control message through a node. 2) the average handle query number in a node.

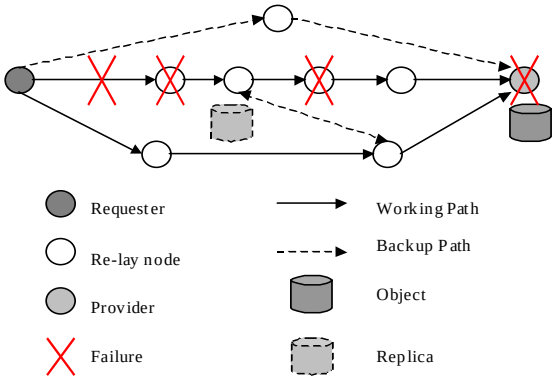


Figure 3 P2P object search

P2P	Scalability	Query Efficiency	Fault Tolerance

1G P2P	Good	Bad	Poor
2G P2P	Fair	Good	Random
3G P2P	Fair	Good	Adversarial Attack

Table 2 Comparison of P2P model

3 P2P Architectures

3.1 P2P Classification

The current P2P networks are so many and each P2P network has so many different properties, that there is no unique classification criterion. We will consider several methods to classify them in order to understand them well.

According to differences of object query mechanism and P2P logical topology, the current P2P architecture can be classified to three types [5]: 1) *Centralized*-All objects index items are kept in a centralized server with such form as <object-key, node-address>. Each arriving node need to actively notify this server about it's kept object information, then nodes only need to consult peer's address from server about it's wanted objects. This type of P2P architecture is very simple and easy to be deployed. But it has the problem of single point of failure, although we can use several parallel servers. The example of the type is *Napster* [6]. 2) *Decentralized but unstructured*-The object query is distributed and the logical P2P topology is often random and unstructured mesh. The query is executed hop-by-hop through this mesh till success/failure or timeout. This type, such as *Gnutella* [7], has no single point failure, but the query efficiency may be low. 3) *Decentralized and structured*-The object query is also distributed, but the logical P2P topology is some structured topology such as *mesh* [8]~[10], *ring* [11], *d-dimension torus* [12], and *butterfly* [13]~[15]. These structured topologies are usually constructed using *distributed hashing table-DHT* techniques, for example [9]~[12]. The object query is also executed hop-by-hop through the structured topology, and is sure to be successful after some deterministic hops under ideal case.

According to the appeared time and purpose, we can classify the current P2P networks into three generations: 1) 1G-first generation: the early P2P networks, such as *Napster* and *Gnutella*, aim to deploy easily and quickly. They are too simple, but have bad scalability or low query efficiency. 2) 2G-second generation: 2G P2P networks usually utilize the *DHT* technique to realize better scalability and query efficiency, and provide load balancing and deterministic search guarantees. But the resilience or fault-tolerance is not good, especially under malicious attacks. 3) 3G-third generation [13]~[15]: the recent proposed P2P network aim to provide high resilience assuming the node will be down with some failure probability. The common used techniques to provide resilience include object replication, expanding the connection number between nodes, some special structured topology. The 2G and 3G P2P networks are often distributed and structured overlay. In the following, we will discuss them in detail respectively according the above two classification methods.

3.2 First Generation of P2P

Napster-CPP [6]
Napster [6], known as music exchange system, and other similar systems have a constantly updated object directory maintained at central *Napster* server(s). Nodes login to this server and send the list of files they can offer, then issue queries to the server to find which other nodes hold their desired files, and finally download the desired objects directly from the object home. Although *Napster's* centralized database naturally avoids query routing and other problems of other P2P systems, it is clear that such centralized approach has single point of failure and very poor scalability. Another feature of this architecture is that it can support partial-match queries (e.g. searching for all objects whose titles contain two or more specific words).

Gnutella-DUPP [7]
There is neither a centralized directory nor any precise control over the network topology or object placement in such architecture as the typical *Gnutella* [7]. *Gnutella* is factually a decentralized file-sharing system whose participants self-organize a virtual mesh network running in a P2P fashion for distributed file search. To participate in *Gnutella*, a node first must connect to a known *Gnutella* node to get lists of some existed *Gnutella* nodes for start-up. To find a

file, a node issues queries to its neighbors. The most typical query method is flooding, where the query is broadcasted to all neighbors within a certain radius or constrained by P2P TTL mechanism. This unstructured architecture is very resilient to nodes entering and leaving the system. However, the current flooding-based lookup mechanism is un-scalable, since it generates large loads on the network participants. Recently, [5] tries to overcome this disadvantage at the best, and proposes two mechanisms: "*dynamic TTL setting or expanding ring*" and "*k-walker random walk*". But "*k-walker random walk*" may have large lookup length (latency). So object replication mechanisms [16][17] (such as uniform replication, proportional replication, square-root proportional replication, and log-form replication) are proposed at the same time to reduce the lookup length. We think lookup message and length can be simultaneously decreased using cache mechanisms such as: cache some objects in the reverse path of queries. Also, *Gnutella* can support partial-match queries.

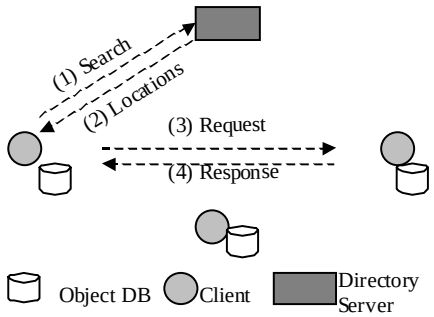


Figure 4 Napster model

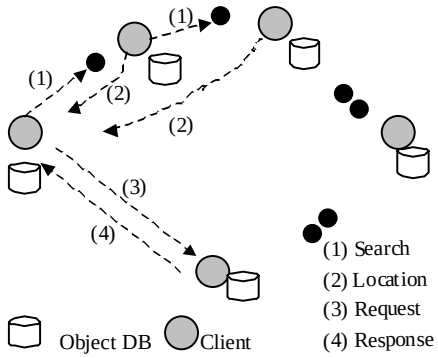


Figure 5 Gnutella model

3.3 Second Generation of P2P

This type of architecture has no central directory server, but has a significant amount of structure. 'Structure" means that the P2P network topology is tightly controlled (such as Mesh [9][10][18], Ring [11] [20-22], d-dimension Torus [12], *K*-ary tree [19], SkipList [23]), and files are placed not at random nodes but at specified locations that will make subsequent queries easier to satisfy. Such structured P2P systems often support a hash-table-like interface, and it is currently quite prevalent in the research literature. It uses precise placement algorithms and specific routing protocols to make the searching efficient. But it is not clear how well such designs work with an extremely transient population of nodes and adversarial node failures (except Butterfly). This type of structured topology supports not partial-match queries, but precise-match queries.

Plaxton [18]

In *Plaxton*, each node or machine can take on the roles of servers (where objects are stored), routers (which forward messages), and clients (origins of requests). In our discussions, we use these terms interchangeably with node. Also, objects and nodes have names independent of their location and semantic properties, in the form of random fixed-length bit-sequences represented by a common base (e.g., 40 Hex digits representing 160 bits). The system assumes entries are roughly evenly distributed in both node and object namespaces, which can be achieved by using the output of hashing algorithms, such as SHA-1(RFC3174). *Plaxton* makes the assumption that the Plaxton mesh is a static data structure, without node or object insertions and deletions.

Object location in *Plaxton* works as follows: 1) A server S1 publishes that it has an object O1 by routing a message to the "root node" of O1. The root node is a unique node in the network used to place the root of the embedded tree for object O1. The publishing process consists of sending toward the root node a message, which contains the mapping <object-id, server-id>. This mapping will be recorded by nodes along the path from the S1 to the root of object O1. 2) During a object location, a query message destined for object O1 is initially routed towards O1's root. 3) At each step, if the message encounters a node that contains the location mapping for O1, it is immediately redirected to the server containing the object O1. Otherwise, the message is forward one step closer to the root. If the message reaches the root, it is guaranteed to find a mapping for the location of O1 if the root of O1 does not fail.

Routing protocol of *Plaxton* works as follows: 1) A node N has a neighbor map with multiple levels, where each level represents a matching suffix up to a digit position in the ID. For example, the 9th entry of the 4th level for node 325AE is the node closest to 325AE in network distance that ends in 95AE. 2) *Plaxton* uses local routing maps (neighbor map) at each node, to incrementally route overlay messages to the destination ID digit by digit (e.g., ***8 => **98 => *598 =>4598 where * represents wildcards). This approach is similar to longest prefix routing in the CIDR IP address allocation architecture.

Tapestry [9]
Tapestry, another self-organizing routing and object location system, is based on *Plaxton*. The core location and routing mechanisms of Tapestry are similar to those of *Plaxton*. But Tapestry goals to improve the capability to detect, circumvent and recover from failures through maintaining periodically updated cached content. It provides the following mechanisms: 1) To detect link and server failures during normal operations, Tapestry can rely on TCP timeouts. Additionally, each Tapestry node uses backpointers to send periodic heartbeats on UDP packets to nodes for which it is a neighbor. By checking the ID of each node a message arrives at, we can quickly detect faulty or corrupted neighbor tables. 2) Tapestry assigns multiple roots to each object. It concatenates a small, globally constant sequence of "salt" values (e.g. 1, 2, 3) to each object ID, then hash the result to identify the appropriate roots. These roots are used during the publishing process to insert location information into the Tapestry. 4) When locating an object, Tapestry performs the same hashing process with the target object ID, generating a set of roots to search. 5) Tapestry also supports some dynamic operations, while static operations in *Plaxton.*

Pastry [10]
Pastry is a location protocol sharing many similarities with Tapestry. Key similarities include the use of prefix/suffix address routing, and similar insertion/deletion algorithms, and similar storage overhead costs. There are several key differences that distinguish Pastry from Tapestry. 1) Objects in Pastry are replicated without controlling by the owner. Upon "publication" of the object, it is replicated and replicas are placed on several nodes whose nodeIDs are closest in the namespace to that of the object's objectID. Tapestry places references to the object location on hops between the server and the root. 2) Pastry assumes that clients use the objectID to attempt to route directly to the vicinity where replicas of the object are kept. While placing actual replicas at different nodes in the network may reduce location latency, it comes at the price of storage overhead at multiple servers, and brings with it a set of questions on security, confidentiality, and consistency.

Chord [11]
Chord is a decentralized P2P lookup service that stores key/value pairs for distributed data items. Given a key, the node responsible for storing the key's value can be determined using a hash function that assigns an identifier to each node and to each key (by hashing the node's IP address and the key). Each key k is stored on the first node whose identifier id is equal or follows k in the identifier space. In Chord, active nodes will form a connected Ring P2P topology under ideal case. Each node maintains a routing table with information for only about $O(\log N)$ nodes. In fact, Chord is similar to binary search, where the searching space will be reduced half after a search/routing-hop. So the number of nodes that must be contacted to resolve a query in a N-node network is $O(\log N)$. As a extent of Chord, if you use K-ary tree search [19], the search hop will decrease to $O(\log_k N)$, while the items of routing table in each node will increase to $O((k-1)* \log_k N)$. There are some other variant of Chord, such as Viceroy [20], Multi-Ring [21][22], etc.

CAN [12]
CAN is another distributed and structured P2P lookup services. Each key will be evenly hashed into a point of d-dimensional space as its identifier. When a node joins, it will randomly select a point of d-dimensional space. Then it will be responsible for half of regions this point belongs to, and hold all keys whose IDs belong to this region. For example, the first arrival node n1 will be responsible for the whole space/region, the second arrival node n2 will split whole region into two parts and takes one of them, and the third arrival node n3 will split the n1 region (if the random point that it selects belongs to this region) or n2 region (otherwise) into two parts and takes one of them also. Each node will keep its neighbor node ID locally, and routing is then performed by forwarding requests to the regions closest to the position of the key. In this way, the expected search length (or P2P hops) is $O(d\sqrt[d]{N})$, and state information kept locally is $O(d)$.

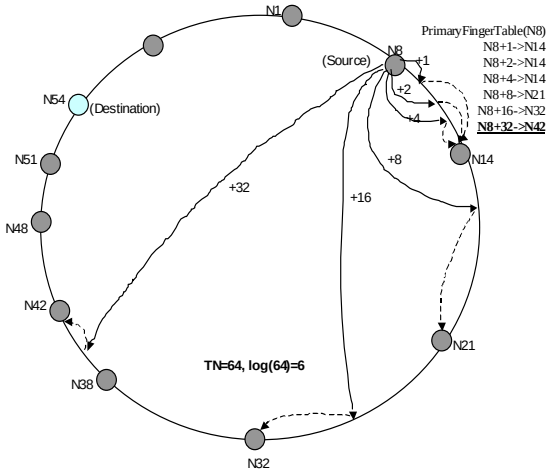
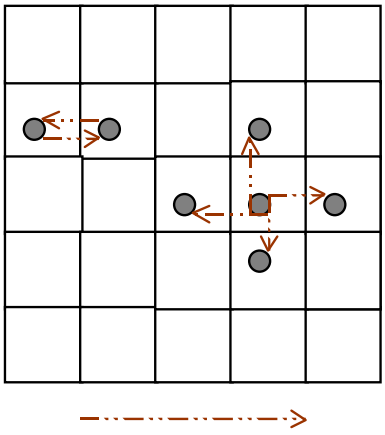


Figure 6 Chord model



Neighboring Connections O(2d=4)

Figure 7 CAN model

3.4 Third Generation of P2P

The first generation P2P is scalable and easy to be deployed, and the second generation P2P based on *DHT* technique has some advantages such as deterministic searching, short query path length, and resistance to random node failure. But they cannot perfectly overcome adversarial attacks. Some researchers try to seek other P2P topology and search algorithm that have good capacity of fault-tolerance. Generally speaking, object replication and multi-path backup can improve the resilience of P2P networks.

Low-Diameter [24]

The main contribution in [24] is to construct a P2P random graph with low diameter *O(logN)* and limited node degree under a probabilistic model of node arrival and departure. The property of low diameter can guarantee short query path length and low control message overhead. Moreover, each node keeps some random connectivity, so can overcome some random attacks. The protocol requires constant memory per node and a ***host server*** with constant memory with which all nodes can connect. The protocol is works as follows:

The ***host server*** maintains a cache of *K* nodes, where *K* is a constant. The host server is reachable by all nodes at all times; however, it need not know of the topology of the network at any time, or even the identities of all nodes currently on the network. A newly arrived node first contacts the host server, which gives it *D* random nodes from the current cache to connect to, thus become into a *d-node*. At some point, the protocol may put a *d-node* into the cache. It stays in the cache until it acquires a total of *C* connections, at which point it leaves the cache, as a *c-node*. A c-node might lose connections after it leaves the cache, but its degree is always at least *D*. This protocol will ensure that the degree (number of neighbors) of all nodes will be in the interval [*D, C+1*], for two constants *D* and *C*.

Assume the arrival of new nodes be modeled by *Poisson* distribution with rate λ, and the duration of time a node stays connected to the network has an exponential distribution with parameter μ. Let G_t be the network at time *t*. Then the P2P topology evolution is the stochastic process $G = (G_t)_{t \geq 0}$. Then this protocol has the following properties: 1) When time *t* is enough bigger, the random graph is connected with probability at least $1 - O(\log^2 N / N)$; 2) When time *t* is enough bigger, the random graph has diameter *O(logN)* with high probability.

Butterfly [13][14]

Butterfly is a censorship resistant P2P system for accessing *N* data items in a network of *N* nodes. It has the following features: 1) Each search for a data item in the network takes *O(logN)* time and requires at most $O(\log^2 N)$ messages. 2) It requires only *O(logN)* memory in each node. 3) Even after adversarial removal of an arbitrarily large constant fraction of the nodes in the system, all but an arbitrarily small fraction of the nodes can find all but an arbitrarily small fraction of the data items. It works as follows: 1) it makes use of butterfly network of depth *logN-loglogN*, the nodes of the butterfly network are called as super-nodes, classified into top super-node, middle super-

node, bottom super-node. 2) Every node randomly chooses *C* top super-nodes, *C* bottom super-nodes, and *C*logN* middle super-nodes to which it will belong, so every super-node is associated with a set of nodes. 3) Every one of the N data items is hashed to *B* random bottom super-nodes using "random oracle model" in [25], and the data item is stored in all the component nodes of all the (bottom) super-nodes to which it has been just hashed to. 4) When performing a search for a data item issued by node n1, lookup message will follow the path from one top super-node of node n1 to one super-node at the bottom level whose index is one of the results from hashing the data item, the operation will repeat until success or until all of paths (from top super-nodes of node n1 to bottom super-node indexed by results from hashing the data item) are completely searched.

Multi-Butterfly [15]
Although the approach in [13-14] can provide resistance to adversarial attacks, yet they need $O(\log^2 N)$ messages per query and $O(\log N)$ object replication factor. Utilizing multi-butterfly topology, MBN improves performance metric to only $O(\log N)$ message per query and $O(1)$ object replication factor.

Multi-butterfly networks were introduced for efficient routing of permutation [39] and have good fault-tolerance [40]. In a *N*-input splitter of a multi-butterfly with multiplicity *d*, every input node is connected to *d* nodes from the upper and lower output nodes. So every output node has edges from *2d* input nodes. In the case of a butterfly network, any pair of input and output nodes is connected by a unique (bit correcting) logical path. However, in the case of a multi-butterfly there is a lot of redundancy since there is a choice of *d* degrees to choose from, at every node, instead of a single edge as in the case of a butterfly. In MBN , the role of all the nodes in a single row of a multi-butterfly is assumed to be played by a single node, thus a corresponding multi-hypercube is formed. Thus a multi-hypercube with n nodes has degree $2d\log N$ for each node, and is a fault-tolerant version of the hypercube network. Theorem 1 and 2 in [15] proves that MBN has the following properties.
-Every node has in-degree and out-degree equal to $d\log N$.
-The data replication factor is $O(1)$.
-Query routing requires no more than $\log N$ hops and no more than $\log N$ messages are sent.
-Even if *f* nodes are deleted (made faulty) by any adversary at least $n-3f/2$ nodes can still reach each at least $n-3f/2$ nodes using $\log N$ length paths.
-MBN can be enhanced so that as long as the number of faults is less than $n/2$, with high probability $(1-\varepsilon)$ fraction of the live nodes can access $(1-\varepsilon)$ fraction of the data items.

4 Object Search Algorithm

Object query algorithm is the key component of P2P networks. It nearly determines the total performance of P2P networks. So we discuss them in a separate section. Object query algorithm can be generally classified as follows: random and deterministic, non-flooding and flooding, centralized and decentralized. Except the centralized query in *Napster*, all the other P2P networks use distributed and hop-by-hop forwarded queries.

4.1 Random query
In [26]~[29], queries are forwarded to a randomly selected neighbor node. In each time, the current node select a unique next-hop randomly from its neighbor set. Random query will improve the query resilience compared to the deterministic unique query, but it will have long query path length. Random query is usually used in unstructured P2P networks such as *Gnutella*.

The probabilistic selection methods include: uniform random selection and proportional random selection according to the connection degree of each neighbor node. Uniform random [29] selection chooses the next-hop randomly from all neighbors with the same probability irrelevant to their connection degree. On the contrary, proportional random [29] selection chooses the next-hop from all neighbors with probability proportional to their connection degree. The neighboring node with bigger connection degree will be more chosen. It is apparent that proportional random selection will have large query hit ratio and better resilience than uniform random selection. But uniform random selection will have better capability of load balancing. Whether is there other ransom selection algorithm to process their advantages simultaneously?

If we randomly choose multiple next-hops, similar to "*k* random walker" in [5], this method can be called as "*k*-random selection" that will make the query path length be decrease and resilience be improved further, but the query message will be multiplied. So there is a tradeoff between unique random query and *k*-random query.

4.2 Non-random query

Compared to random query, non-random query choose the next-hop deterministically, which is usually used in structured P2P networks such as 2G *DHT*-based P2P networks. Non-random query can also be classified into unique-query and multi-query, flooding-based query is a special form of multiple-query.

In 2G *DHT*-based P2P networks, the query algorithm is unique query and one of greedy algorithm [30], because the chosen next-hop is the nearest node to the destination. This type of algorithm guarantees to find the destination if all the nodes are active, but the query hit ratio will be very slow if there are some attack or failure nodes. The average query path length and query message of this type query are usually *O(logN)*. Also, it supports load balancing.

Another unique non-random query is non-greedy algorithm. It always selects the next-hop with maximal connection degree [31], in order to guarantee short query path length and high query hit ratio if there are node failure. This method is suitable to unstructured P2P network. The extra requirement is to keep the connection degree of all the neighboring nodes.

Multi-query will select several next-hops according deterministic methods, such as the *breadth-first search* used in *Gnutella*. This type of query has good query hit ratio and query path length, but with large query message.

5 P2P Modeling

The purpose of P2P modeling is to analyze its performance and other behaviors in ideal and heterogeneous environment. There are many factors to be considered before carrying out analysis: node failure model, object-requesting model, P2P topology, search algorithm, object replication model, and query arrival model. Then according to these assumed models, we can analyze the P2P performance such as: query hit ratio, query path length, query message number, and node process load. In analysis, the key is to compute the query forwarding probability, $p_{i,j}^{k}$, the probability that the query for object *k* is forwarded from node *i* to node *j*.

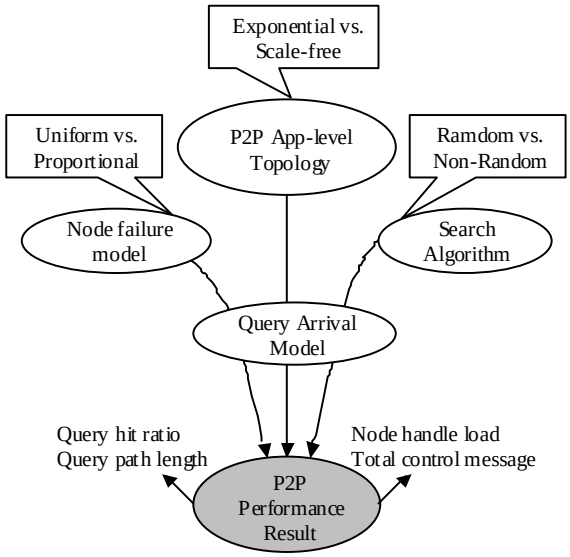


Figure 8 P2P Modeling Factors

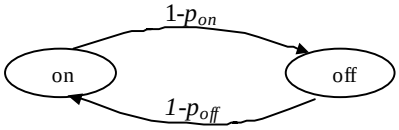


Figure 9 Node failure model

Node failure model [32]

The node will fails under some cases. It is common to use two states on-off model to describe this phenomenon (figure 9). Each node will be in on state with probability p_{on}, and in off state with probability p_{off}. Another model is that nodes arrive with rate of Poisson distribution with parameter λ, and the stay time in system is exponential distribution with parameter μ.

Object-requesting model

It is often assumed that each the requesting for each object *i* is Poisson distribution with average requesting rate r_i. So the query handling can be seen as M/G/1 model, even M/D/1 queuing model assuming each object with the same handling time.

P2P topology

P2P topology can be classified into two types: exponential topology or scale-free topology. But we can describe it using adjacency matrix *A*, which can tell us the simple path number of some length for any pair of two nodes.

Object replication model

We can simply assume that each object *i* has c_i replications, which uniformly distributes in total P2P nodes. Replication mechanism can improve P2P *resilience* and *query efficiency*. After obtaining performance metric dependent on replication number, an optimal replication policy can be formalized for certain optimization objects [16][17]. These optimization problems can give out the optimal replication number. Paper [16] has presented three replication policies for unstructured P2P networks, but it does not consider P2P node failure model.

Object search algorithm

Object search algorithm is the key component of any P2P networks. Combing P2P adjacency matrix *A* and object search algorithm, we can determine $p_{i,j}^k$, the probability that the query for object *k* is forwarded to the next hop *j* from the current node *i*. After obtain $p_{i,j}^k$, we can easily compute the query hit ratio, query path length, and some other metrics using the methods in [33].

There are some P2P modeling results in [32-36]. Paper [32][34][35] analyzes the performance of structured P2P networks including Chord, CAN, Tapestry, and Pastry, through constructing a Markov-model for their query methods while omitting the "back-tracking mechanism". Also, they do not consider replication policy and do not compute the each P2P node process load.

Paper [33] gives out a framework based on Markov-chain, to analyze some routing and object search protocol, but it does not consider the node failure model and does not compute the query hit ratio. Although this method is not proposed specially for P2P networks, yet we can use it to analyze P2P performance combining the methods in [32].

6 P2P Applications

7 Future Directions

There are some problems and future directions about P2P overlay networks: 1) QoS problems [37][38] in P2P networks. The single-hop in P2P layer maybe maps into multi-hops in IP layer, so the query delay and object download delay probably much longer even if the query path length in P2P layer is very small. 2) The congestion problem in P2P network. If there are some hot objects, then the query and download related to these hot objects

will make some node congested [33], even block the whole P2P networks. One of solution is object replication and search algorithm capable of load balancing. 3) The resilience of P2P networks. Paper [29] compares resilience of two types of topology, and shows that: the scale-free topology (such as *Gnutella*) has better resilience for random failure, but worse for attacks than exponential topology (such as 2G-P2P). Recently, Butterfly-based P2P network are proposed to overcome failure and attacks. They need much more routing table item or object replications, and the capability to overcome attacks is also unknown. In short, although some current P2P networks has good capability to overcome random failure, it is unknown these network can overcome the special attacks. 4) P2P performance modeling. There are some theoretical performance analysis results in [32]~[36]. But they do not consider the complete factors listed in section 5. Also, there is no any good close-form solution of P2P performance. 5) Hybrid P2P architecture. Besides the existed hierarchy P2P networks [41][42], we can continue to seek some advanced hierarchical P2P networks, which has advantages of scalability, manageability, and resilience for large-scale applications.

8 Conclusions

P2P network is factually an overlay network for distributed object store, search and sharing. This paper will present a survey about it's recent. First, we will introduce a simple definition of P2P, and define some common operation procedures in P2P. Second, P2P performance metric will be discussed in order to understand and differentiated the practical P2P architectures and protocols. Third, the current P2P architecture will be classified and compared in detail. Then, object search protocols will be carefully discussed with corresponding P2P architecture. We also review the current P2P modeling outputs and list the general modeling approaches. Finally, several novel applications based on P2P techniques and future research directions of P2P overlay networks are briefly investigated. This paper aims to give a clear and complete survey of P2P networks according to clues of its development and evolution process.

References

[1]. Oram, editor, "Peer-to-peer: Harnessing the power of disruptive technologies," *O'Reilly & Associates*, March 2001. [online]: http://www.oreilly.de/catalog/peertopeer/

[2]. K.Kant, R.Iyer, and V.Tewari, "A Framework for classifying peer-to-peer technologies," in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002.

[3]. K.Aberer, M.Hauswirth, "An overview on peer-to-peer information systems," to be published in Proceedings of *Workshop on Distributed Data and Structures (WDAS-2002)*, Paris, France, 2002. [online]: http://lsirpeople.epfl.ch/hauswirth/

[4]. D.S.Milojicic, et al., "Peer-to-peer computing", *HL Laboratories Research Report*, 2002. [online]: http://www.hpl.hp.com/techreports/2002/HPL-2002-57.html

[5]. Q.Lv, P.Cao, E.Cohen, K.Li, and S.Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proceedings of 16th ACM International Conference on Supercomputing (ICS'02)*, New York, USA, June 2002.

[6]. Napster: http://www.napster.com/

[7]. Gnutella: http://www.gnutella.com/

[8]. I.Clarke, T.W.Hong, S.G.Miller, O.Sandberg, and B.Wiley, "Protecting free expression online with Freenet," *IEEE Internet Computing* 6(1), pp40-49 2002.

[9]. B.Y.Zhao, J.Kubiatowicz, and A.Joseph. "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Technical Report UCB/CSD-01-1141*, University of California at Berkeley, Computer Science Department, 2001. [online]: http://www.cs.berkeley.edu/~ravenben/tapestry/html/papers.html

[10]. A.Rowstron and P.Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[11]. I.Stoica, R.Morris, D.Karger, F.Kaashoek, and H.Balakrishnan, "Chord: A scalable peer-topeer lookup service for Internet applications," in *Proceedings of ACM SIGCOMM'2001*, August 2001. [online]: http://www.pdos.lcs.mit.edu/chord/#pubs

[12]. S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker, "A scalable content addressable network," in *Proceedings of ACM SIGCOMM'2001*, August 2001. [online]: http://www.icir.org/sylvia/

[13]. A.Fiat and J.Saia, "Censorship resistant peer-to-peer content addressable networks," in *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, USA, January 2002. [online] http://www.cs.unm.edu/~saia/

[14]. J.Saia, A.Fiat, S.Gribble, A.R.Karlin, and S.Saroiu, "Dynamically fault-tolerant content addressable networks," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002. [online]: http://www.cs.rice.edu/Conferences/IPTPS02/

[15]. M.Datar, "Butterflies and peer-to-peer networks, " *Technial Report*, January 2002. [online]: http://dbpubs.stanford.edu:8090/pub/2002-5

[16]. E.Cohen and S.Shenker, "Replication strategies in unstructured peer-to-peer networks," in *Proceedings of ACM SIGCOMM'02*, Aug. 2002.

[17]. J.Kangasharju, K.W.Ross, and D.A.Turner, "Optimal content replication in peer-to-peer communities," in *Submission*. [online]: http://www.eurecom.fr/~kangasha/

[18]. C.Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *Proceedings of ACM SPAA. ACM*, June 1997. [online]: http://www.public.asu.edu/~aricha/publs.html.

[19]. S.E.Ansary, L.O.Alima, P.Brand, and S.Haridi, "A framework for peer-to-peer lookup services based on k-ary search," *SICS technical report T2002-06*, 2002

[20]. D.Malkhi, M.Naor, and D.Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly", in *Proceedings of PODC'2002*, 2002.

[21]. M.Junginger, and Y.Lee, "The multi-ring technology-High performance group communication in peer-to-peer networks," in *Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P'02)*, 2002.

[22]. J.Considine and T.Florio, "Scalable peer-to-peer indexing with constant state," *Technical Report*, September 2002

[23]. N.J.A.Harvey, M.B.Jones, S.Saroiu, M.Theimer, and A.Wolman, "SkipNet: A scalable overlay network with practical locality properties," Accepted for publication to the *Fourth USENIX Symposium on Internet Technologies and Systems (USITS'03)*, September 2002.

[24]. G.Pandurangan, P.Raghavan, and E.Upfal, "Building low-diameter P2P networks," in *Proceedings of IEEE Symposium on Foundations of Computer Science 2001*, 2001
[25]. M.Bellare and P.Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62-73, 1993. [online]: http://www.cs.ucdavis.edu/~rogaway/papers/ro-abstract.html
[26]. L.Barrière, P.Fraigniaud, E.Kranakis, and Danny Krizanc, "Efficient routing in networks with long range contacts," in *Proceedings of 15th International Conference, DISC 2001*, Lisbon, Portugal, October 3-5, 2001.
[27]. J.Kleinberg, "The small-world phenomenon: An algorithmic perspective" in *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2002.
[28]. S.C.Rhea and J.Kubiatowicz, "Probabilistic location and routing," in *Proceedings of IEEE INFOCOM'2002*, 2002.
[29]. B.I.Kim, C.N.Yoon, S.K.Han, and H.Jeong, "Path finding strategies in scale-free networks," *Physical Review E*, vol.65, January 23, 2002.
[30]. J.Aspnes, Z.Diamadi, and G.Shah, "Fault-tolerant routing in peer-to-peer systems," in *Proceedings of PODC'2002*, 2002.
[31]. L.A.Adamic, R.M.Lukose, A.R.Puniyani, and B.A.Huberman, "Search in power-law networks," *Physical Review E*, vol.64, 2001.
[32]. S.Wang and D. Xuan, "A Markov-chain based approach to resilience of structured P2P systems under failures and attacks". *Technical Report*, 2002.
[33]. R.Guimera,A.D.Guilera, F.V.Redondo, A.Cabrales, and A.Arenas, "Optimal network topologies for local search with congestion," *Physical Review Letter*, vol.89, no.24, December 9, 2002.
[34]. D.Xuan, M.Krishnamoorthy, and S.Wang, "Analyzing and enhancing the resilience of peer-to-peer systems to routing attack," *Technical Report*, 2002.
[35]. S.Wang, M.Krishnamoorthy and D.Xuan, "Analyzing resilience of structured peer-to-peer systems," *Technical Report*, 2002.
[36]. H.C.Hsiao and C.T.King, "Modeling and evaluating peer-to-peer storage architectures," in *Proceedings of International Parallel and Distributed Processing Symposium'2002 (IPDPS'02)*, April 2002.
[37]. S.Haridi, "NetProbe: A component for enhancing efficiency of overlay networks in P2P systems," in *Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing Systems*, Linköping, Sweden, September 5-7, 2002.
[38]. D.G.Andersen, "Resilient Overlay Networks," *Masters Thesis*, Massachusetts Institute of Technology, 2001.
[39]. E.Upfal, "An $O(\log N)$ deterministic packet-routing scheme," in *Journal of ACM*, vol.39, no.1, pp.55-70, January 1992.
[40]. T.Leighton, and B.Maggs, "Expanders might be practical: Fast algorithms for routing around faults on multi-butterflies," in *Proceedings of 30th IEEE symposium on Foundations of Computer Science*, pp.384-389, Los Alamitos, USA, 1989.
[41]. B.Yang, and H.G.Molina, "Designing a super-node network, " in *Proceedings of ICDE*, 2003.

Appendix: Tables

P2P	Query path length	Query control message	Routing table item	Resilience	1G-P2P			2G-P2P	3G-P2P
					CPP	DUPP	DSPP	DSPP	DSPP
Napster[6]	*O(1)*	1	$O(N)$	*SF**	√				
Gnutella[7]	*<TTL*	$O(D^{TTL})$	$O(D)$	*Fair*		√			
Freenet[8]	*<TTL*	*O(TTL)*	$O(D)$	*Fair*			√		
Chord[11]	$O(\log N)$	$O(\log N)$	$O(\log N)$	*Poor*				√	
CAN[12]	$O(d\sqrt[d]{N})$	$O(d\sqrt[d]{N})$	$O(d)$	*?*				√	
Tapestry[9]	$O(\log N)$	$O(\log N)$	$O(\log N)$	*Fair*				√	
Pastry[10]	$O(\log N)$	$O(\log N)$	$O(\log N)$	*Fair*				√	
Viceroy[20]	$O(\log N)$	$O(\log N)$	*7*	*?*				√	
BF[13]	*O(logN)*	*O($\log^2 N$)*	*O(logN)*	*Good*					√
DBF[14]	*O(logN)*	*O($\log^3 N$)*	*O(logN)*	*Good*					√
MBF[15]	*O(logN)*	*O(logN)*	*O(logN)*	*Good*					√

Table 2 P2P networks comparison (*: single point of failure)

Algorithm	Hit ratio	Path length	Load Balance	Used in	Random		Non-Random	
					Unique	Multiple	Unique	Multiple
Uniform Random [29]	*Fair*	*O($\log^2 N$)*	*Yes*	*DUPP*	√			
Proportional Random [29]	*Fair*	*O($\log^2 N$)*	*Yes*	*DUPP*	√			
Small World Greedy [27]	*Fair*	*O($\log^2 N$)*	*Yes*	*DUPP*	√			
k-Walker [5]	*Good*	*<TTL*	*Yes*	*DUPP*		√		
Maximal Degree First [31]	*Fair*	---	*No*	*DUPP*			√	
Gnutella (BFS)[7]	*Good*	*<TTL*	*Yes*	*DUPP*				√
Freenet (DFS)[8]	*Fair*	*<TTL*	*Yes*	*DSPP*			√	
2&3G P2P (Greedy)	*Good*	*O(logN)*	*Yes*	*DSPP*			√	

Table 3 Distributed object search algorithm