A Survey and Comparison of Peer-to-Peer Overlay Network Schemes

Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim

***Abstract*— Over the Internet today, computing and communications environments are significantly more complex and chaotic than classical distributed systems, lacking any centralized organization or hierarchical control. There has been much interest in emerging Peer-to-Peer (P2P) network overlays because they provide a good substrate for creating large-scale data sharing, content distribution and application-level multicast applications. These P2P networks try to provide a long list of features such as: selection of nearby peers, redundant storage, efficient search/location of data items, data permanence or guarantees, hierarchical naming, trust and authentication, and, anonymity. P2P networks potentially offer an efficient routing architecture that is self-organizing, massively scalable, and robust in the wide-area, combining fault tolerance, load balancing and explicit notion of locality. In this paper, we present a survey and comparison of various Structured and Unstructured P2P networks. We categorize the various schemes into these two groups in the design spectrum and discuss the application-level network performance of each group.**

***Index Terms*— Peer-to-Peer, Distributed Scalable Algorithms, Lookup Protocols, Overlay Routing, Overlay Networks.**

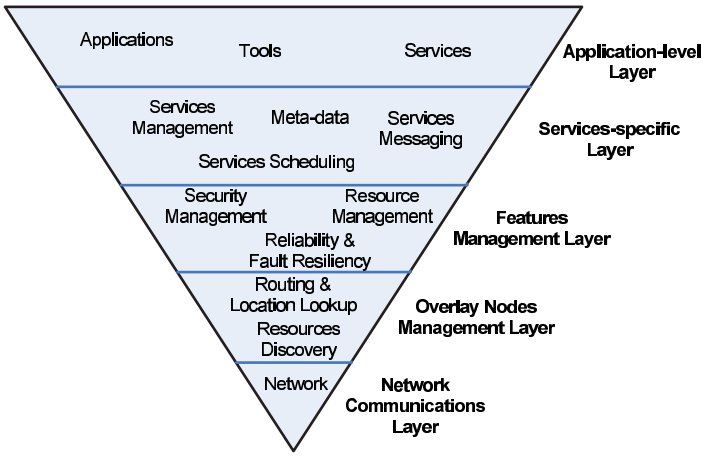

Fig. 1. An Abstract P2P Overlay Network Architecture

I. INTRODUCTION

PEER-TO-PEER (P2P) overlay networks are distributed systems in nature, without any hierarchical organization or centralized control. Peers form self-organizing overlay networks that are overlayed on the Internet Protocol (IP) networks, offering a mix of various features such as robust wide-area routing architecture, efficient search of data items, selection of nearby peers, redundant storage, permanence, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance. Peer-to-peer overlay systems go beyond services offered by client-server systems by having symmetry in roles where a client may also be a server. It allows access to its resources by other systems and supports resource-sharing, which requires fault-tolerance, self-organization and massive scalability properties. Unlike Grid systems, P2P overlay networks do not arise from the collaboration between established and connected groups of systems and without a more reliable set of resources to share.

We can view P2P overlay network models spanning a wide spectrum of the communication framework, which specifies a fully-distributed, cooperative network design with peers building a self-organizing system. Figure 1 shows an abstract P2P overlay architecture, illustrating the components in the overlay communications framework. The Network Communications layer describes the network characteristics of desktop machines connected over the Internet or small wireless or sensor-based devices that are connected in an ad-hoc manner. The dynamic nature of peers poses challenges in communication paradigm. The Overlay Nodes Management layer covers the management of peers, which include discovery of peers and routing algorithms for optimization. The Features Management layer deals with the security, reliability, fault resiliency and aggregated resource availability aspects of maintaining the robustness of P2P systems. The Services Specific layer supports the underlying P2P infrastructure and the application-specific components through scheduling of parallel and computation-intensive tasks, content and file management. Meta-data describes the content stored across the P2P peers and the location information. The Application-level layer is concerned with tools, applications and services that are implemented with specific functionalities on top of the underlying P2P overlay infrastructure. So, there are two classes of P2P overlay networks: *Structured* and *Unstructured*.

The technical meaning of *Structured* is that the P2P overlay network topology is tightly controlled and content are placed not at random peers but at specified locations that will make subsequent queries more efficient. Such Structured P2P systems use the Distributed Hash Table (DHT) as a substrate, in which data object (or value) location information is placed deterministically, at the peers with identifiers corresponding to the data object's unique *key*. DHT-based systems have a

Manuscript received March 31, 2004; revised November 20, 2004.

Eng Keong Lua, Jon Crowcroft and Marcelo Pias are with the University of Cambridge, Computer Laboratory.

Ravi Sharma is with the Nanyang Technological University.

Steven Lim is with the Microsoft Asia.

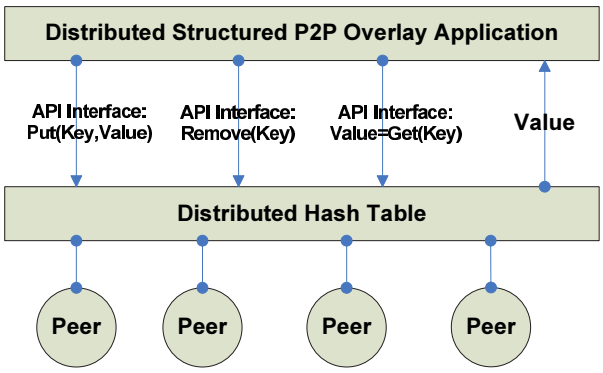

Fig. 2. Application Interface for Structured DHT-based P2P Overlay Systems

property that consistently assigned uniform random NodeIDs to the set of peers into a large space of identifiers. Data objects are assigned unique identifiers called *keys*, chosen from the same identifier space. *Keys* are mapped by the overlay network protocol to a unique live peer in the overlay network. The P2P overlay networks support the scalable storage and retrieval of {**key,value**} pairs on the overlay network, as illustrated in Figure 2. Given a *key*, a store operation *(put(key,value))* lookup retrieval operation *(value=get(key))* can be invoked to store and retrieve the data object corresponding to the *key*, which involves routing requests to the peer corresponding to the *key*.

Each peer maintains a small routing table consisting of its neighboring peers' NodeIDs and IP addresses. Lookup queries or message routing are forwarded across overlay paths to peers in a progressive manner, with the NodeIDs that are *closer* to the *key* in the identifier space. Different DHT-based systems will have *different* organization schemes for the data objects and its *key* space and routing strategies. In theory, DHT-based systems can guarantee that any data object can be located in a small $O(logN)$ overlay hops on average, where N is the number of peers in the system. The underlying network path between two peers can be significantly different from the path on the DHT-based overlay network. Therefore, the lookup latency in DHT-based P2P overlay networks can be quite high and could adversely affect the performance of the applications running over it. Plaxton et al. [1] provides an elegant algorithm that achieves nearly optimal latency on graphs that exhibit power-law expansion [2], at the same time, preserving the scalable routing properties of the DHT-based system. However, this algorithm requires pair-wise probing between peers to determine latencies and it is unlikely to scale to a large number of peers in the overlay. DHT-based systems [3]–[7] are an important class of P2P routing infrastructures. They support the rapid development of a wide variety of Internet-scale applications ranging from distributed file and naming systems to application-layer multicast. They also enable scalable, wide-area retrieval of shared information.

In 1999, the Napster [8] pioneered the idea of a peer-to-peer file sharing system supporting a centralized file search facility. It was the first system to recognize that requests for popular content need not to be sent to a central server but instead it could be handled by many peers, that have the requested content. Such P2P file-sharing systems are self-scaling in that as more peers join the system, they add to the aggregate download capability. Napster achieved this self-scaling behavior by using a centralized search facility based on file lists provided by each peer, thus, it does not require much bandwidth for the centralized search. Such a system has the issue of a single point of failure due to the centralized search mechanism. However, a lawsuit filed by the Recording Industry Association of America (RIAA) forced Napster to shut down the file-sharing service of digital music — literally, its *killer application*. However, the paradigm caught the imagination of platform providers and users alike. Gnutella [9]–[11] is a decentralized system that distributes both the search and download capabilities, establishing an overlay network of peers. It is the first system that makes use of an Unstructured P2P overlay network. An Unstructured P2P system is composed of peers joining the network with some loose rules, without any prior knowledge of the topology. The network uses flooding as the mechanism to send queries across the overlay with a limited scope. When a peer receives the flood query, it sends a list of all content matching the query to the originating peer. While flooding-based techniques are effective for locating highly replicated items and are resilient to peers joining and leaving the system, they are poorly suited for locating rare items. Clearly this approach is not scalable as the load on each peer grows linearly with the total number of queries and the system size. Thus, Unstructured P2P networks face one basic problem: peers readily become overloaded, therefore, the system does not scale when handling a high rate of aggregate queries and sudden increase in system size.

Although Structured P2P networks can efficiently locate rare items since the key-based routing is scalable, they incur significantly higher overheads than Unstructured P2P networks for popular content. Consequently, over the Internet today, the decentralized Unstructured P2P overlay networks are more commonly used. However, there are recent efforts on Key-based Routing (KBR) API abstractions [12] that allow more application-specific functionality to be built over this common basic KBR API abstractions, and OpenHash (Open publicly accessible DHT service) [13] that allows the unification platform of providing developers with basic DHT service models that runs on a set of infrastructure hosts, to deploy DHT-based overlay applications without the burden of maintaining a DHT and with ease of use to spur the deployment of DHT-based applications. In contrast, Unstructured P2P overlay systems are Ad-Hoc in nature, and do not present the possibilities of being unified under a common platform for application development.

In the sections II and IV of the paper, we will describe the key features Structured P2P and Unstructured P2P overlay networks and their operation functionalities. After providing a basic understanding of the various overlays schemes in these *two* classes, we proceed to evaluate these various overlays schemes in both classes and discuss its developments in sections III and V. Then, we attempt to use the taxonomy to make comparisons between the various discussed Structured

and Unstructured P2P overlay schemes:

- *Decentralization* — examine whether the overlay system is distributed.
- *Architecture* — describe the overlay system architecture with respect to its operation.
- *Lookup Protocol* — the lookup query protocol adopted by the overlay system.
- *System Parameters* — the required system parameters for the overlay system operation.
- *Routing Performance* — the lookup routing protocol performance in overlay routing.
- *Routing State* — the routing state and scalability of the overlay system.
- *Peers Join and Leave* — describe the behavior of the overlay system when churn and self-organization occurred.
- *Security* — look into the security vulnerabilities of overlay system.
- *Reliability and Fault Resiliency* — examine how robust the overlay system when subjected to faults.

Lastly, in section VI, we conclude with some thoughts on the relative applicability of each class to some of the research problems that arise in Ad-Hoc, location-based or content delivery networks.

II. STRUCTURED P2P OVERLAY NETWORKS

In this category, the overlay network assigns keys to data items and organizes its peers into a graph that maps each data key to a peer. This structured graph enables efficient discovery of data items using the given keys. However, in its simple form, this class of systems does not support complex queries and it is necessary to store a copy or a pointer to each data object (or value) at the peer responsible for the data object's key. In this section, we survey and compare the *Structured* P2P overlay networks: Content Addressable Network (CAN) [5], Tapestry [7], Chord [6], Pastry [4], Kademlia [14] and Viceroy [15].

A. Content Addressable Network (CAN)

The Content Addressable Network (CAN) [5] is a distributed decentralized P2P infrastructure that provides hash-table functionality on Internet-like scale. CAN is designed to be scalable, fault-tolerant, and self-organizing. The architectural design is a virtual multi-dimensional Cartesian coordinate space on a multi-torus. This d-dimensional coordinate space is completely logical. The entire coordinate space is dynamically partitioned among all the peers (N number of peers) in the system such that every peer possesses its individual, distinct zone within the overall space. A CAN peer maintains a routing table that holds the IP address and virtual coordinate zone of each of its neighbors in the coordinate space. A CAN message includes the destination coordinates. Using the neighbor coordinates, a peer routes a message towards its destination using a simple greedy forwarding to the neighbor peer that is closest to the destination coordinates. CAN has a routing performance of $O(d.N^{\frac{1}{d}})$ and its routing state is of $2.d$ bound. As shown in Figure 3 which we adapted from the

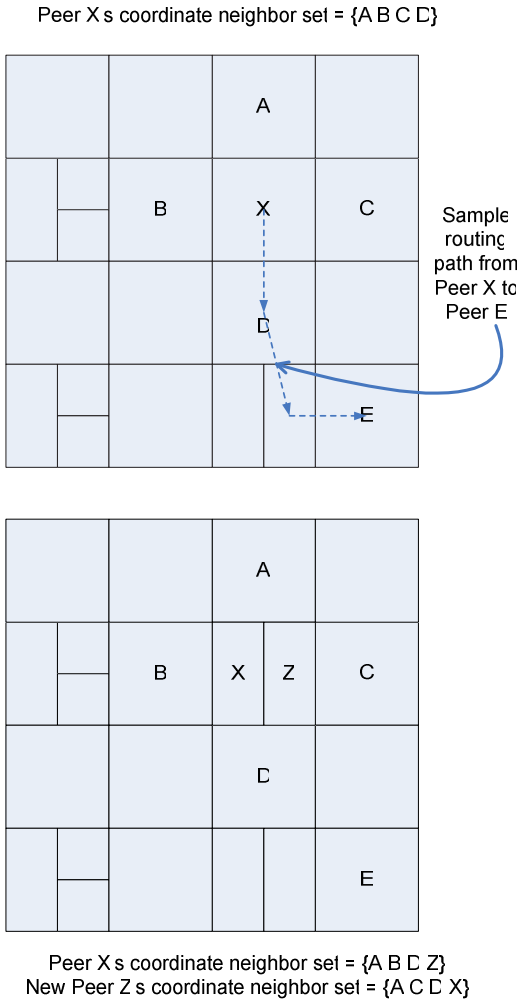

Fig. 3. Example of 2-d space CAN before and after Peer Z joins

CAN paper [5], the virtual coordinate space is used to store {**key,value**} pairs as follows: to store a pair {**K,V**}, **key K** is deterministically mapped onto a point P in the coordinate space using a uniform hash function. The lookup protocol to retrieve an entry corresponding to key K, any peer can apply the same deterministic hash function to map K onto point P and then retrieve the corresponding **value V** from the point P. If the requesting peer or its immediate neighbors do not own the point P, the request must be routed through the CAN infrastructure until it reaches the peer where P lays. A peer maintains the IP addresses of those peers that hold coordinate zones adjoining its zone. This set of immediate neighbors in the coordinate space serves as a coordinate routing table that enables efficient routing between points in this space.

A new peer that joins the system must have its own portion of the coordinate space allocated. This can be achieved by splitting existing peer's zone in half; retaining half for the peer and allocating the other half to the new peer. CAN has an associated DNS domain name which is resolved into IP address of one or more CAN bootstrap peers (which maintains a partial list of CAN peers). For a new peer to join CAN

network, the peer looks up in the DNS a CAN domain name to retrieve a bootstrap peer's IP address, similar to the bootstrap mechanism in [16]. The bootstrap peer supplies the IP addresses of some randomly chosen peers in the system. The new peer randomly chooses a point P and sends a JOIN request destined for point P. Each CAN's peer uses the CAN routing mechanism to forward the message until it reaches the peer in which zone P lies. The current peer in zone P then splits its in half and assigns the other half to the new peer. For example, in a 2-dimensional $(2-d)$ space, a zone would first be split along the X dimension, then the Y, and so on. The {K,V} pairs from the half zone to be handed over are also transferred to the new peer. After obtaining its zone, the new peer learns of the IP addresses of its neighbor set from the previous peer in point P, and adds to that previous peer itself.

When a peer leaves the CAN network, an immediate takeover algorithm ensures that one of the failed peer's neighbors takes over the zone and starts a takeover timer. The peer updates its neighbor set to eliminate those peers that are no longer its neighbors. Every peer in the system then sends soft-state updates to ensure that all of their neighbors will learn about the change and update their own neighbor sets. The number of neighbors a peer maintains depends only on the dimensionality of the coordinate space (i.e. $2.d$) and it is independent of the total number of peers in the system.

The Figure 3 example illustrated a simple routing path from peer X to point E and a new peer Z joining the CAN network. For a d-dimensional space partitioned into n equal zones, the average routing path length is $(d/4)$ x $(n^{\frac{1}{d}})$ hops and individual peers maintain a list of $2.d$ neighbors. Thus, the growth of peers (or zones) can be achieved without increasing per peer state while the average path length grows as $O(n^{\frac{1}{d}})$. Since there are many different paths between two points in the space, when one or more of a peer's neighbors fail, this peer can still route along the next best available path.

Improvement to the CAN algorithm can be done by maintaining multiple, independent coordinate spaces with each peer in the system being assigned a different zone in each coordinate space, called *reality*. For a CAN with r realities, a single peer is assigned r coordinate zones, one on each reality available, and this peer holds r independent neighbor sets. The contents of the hash table are replicated on every reality, thus improving data availability. For further data availability improvement, CAN could use k different hash functions to map a given key onto k points in the coordinate space. This results in the replication of a single {key,value} pair at k distinct peers in the system. A {key,value} pair is then unavailable only when all the k replicas are simultaneously unavailable. Thus, queries for a particular hash table entry could be forwarded to all k peers in parallel thereby reducing the average query latency, and reliability and fault resiliency properties are enhanced.

CAN could be used in large scale storage management systems such as the OceanStore [17], Farsite [18], and Publius [19]. These systems require efficient insert and retrieval of content in a large distributed storage network with a scalable indexing mechanism. Another potential application for CANs is in the construction of wide-area name resolution services that decouple the naming scheme from the name resolution process. This enables an arbitrary and location-independent naming scheme.

B. Chord

Chord [6] uses consistent hashing [20] to assign keys to its peers. Consistent hashing is designed to let peers enter and leave the network with minimal interruption. This decentralized scheme tends to balance the load on the system, since each peer receives roughly the same number of keys, and there is little movement of keys when peers join and leave the system. In a steady state, for N peers in the system, each peer maintains routing state information for about only $O(logN)$ other peers (N number of peers in the system). This may be efficient but performance degrades gracefully when that information is out-of-date.

The consistent hash functions assign peers and data keys an m-bit identifier using SHA-1 [21] as the base hash function. A peer's identifier is chosen by hashing the peer's IP address, while a key identifier is produced by hashing the data key. The length of the identifier m must be large enough to make the probability of keys hashing to the same identifier negligible. Identifiers are ordered on an identifier circle modulo $2m$. Key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. This peer is called the successor peer of key k, denoted by $successor(k)$. If identifiers are represented as a circle of numbers from 0 to $2m-1$, then $successor(k)$ is the first peer clockwise from k. The identifier circle is termed as the Chord ring. To maintain consistent hashing mapping when a peer n joins the network, certain keys previously assigned to n's successor now need to be reassigned to n. When peer n leaves the Chord system, all of its assigned keys are reassigned to n's successor. Therefore, peers join and leave the system with $(logN)^2$ performance. No other changes of keys assignment to peers need to occur. In Figure 4 (adapted from [6]), the Chord ring is depicted with $m=6$. This particular ring has ten peers and stores five keys. The successor of the identifier 10 is peer 14, so key 10 will be located at NodeID 14. Similarly, if a peer were to join with identifier 26, it would store the key with identifier 24 from the peer with identifier 32.

Each peer in the Chord ring needs to know how to contact its current successor peer on the identifier circle. Lookup queries involve the matching of key and NodeID. For a given identifier could be passed around the circle via these successor pointers until they encounter a pair of peers that include the desired identifier; the second peer in the pair is the peer the query maps to. An example is presented in Figure 4, whereby peer 8 performs a lookup for key 54. Peer 8 invokes the find_successor operation for this key, which eventually returns the successor of that key, i.e. peer 56. The query visits every peer on the circle between peer 8 and peer 56. The response is returned along the reverse of the path.

As m is the number of bits in the key/NodeID space, each peer n maintains a routing table with up to m entries, called the finger table. The i^{th} entry in the table at peer n contains

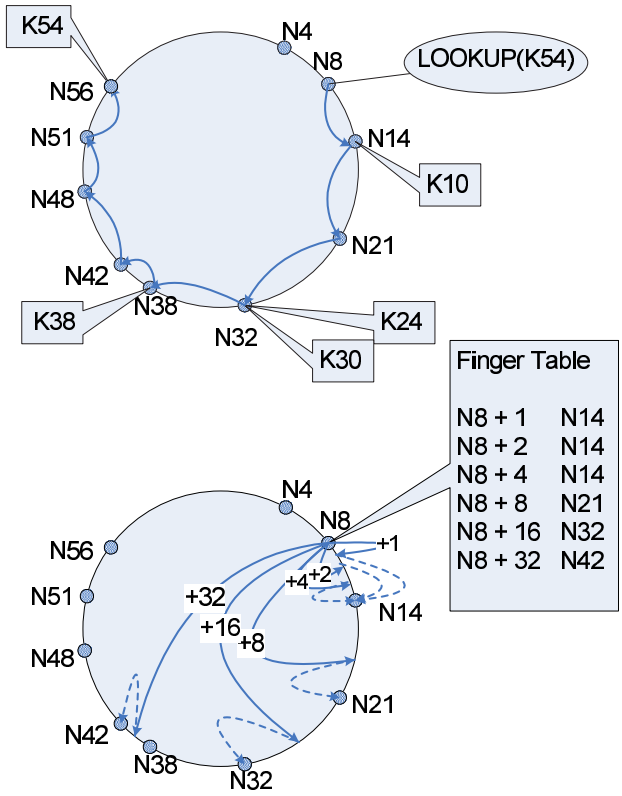

Fig. 4. Chord ring with identifier circle consisting of ten peers and five data keys. It shows the path followed by a query originated at peer 8 for the lookup of key 54. Finger table entries for peer 8.

the identity of the first peer s that succeeds n by at least 2^{i-1} on the identifier circle, i.e. $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$. Peer s is the i^{th} finger of peer n $(n.finger[i])$. A finger table entry includes both the Chord identifier and the IP address (and port number) of the relevant peer. Figure 4 shows the finger table of peer 8, and the first finger entry for this peer points to peer 14, as the latter is the first peer that succeeds $(8+20)$ mod $26 = 9$. Similarly, the last finger of peer 8 points to peer 42, i.e. the first peer that succeeds $(8 + 25)$ mod $26 = 40$. In this way, peers store information about only a small number of other peers, and know more about peers closely following it on the identifier circle than other peers. Also, a peer's finger table does not contain enough information to directly determine the successor of an arbitrary key k. For example, peer 8 cannot determine the successor of key 34 by itself, as successor of this key (peer 38) is not present in peer 8's finger table.

When a peer joins the system, the successor pointers of some peers need to be changed. It is important that the successor pointers are up to date at any time because the correctness of lookups is not guaranteed otherwise. The Chord protocol uses a *stabilization* protocol [6] running periodically in the background to update the successor pointers and the entries in the finger table. The correctness of the Chord protocol relies on the fact that each peer is aware of its successors. When peers fail, it is possible that a peer does not know its new successor, and that it has no chance to learn about it. To avoid this situation, peers maintain a successor list of size r, which contains the peer's first r successors. When the successor peer does not respond, the peer simply contacts the next peer on its successor list. Assuming that peer failures occur with a probability p, the probability that every peer on the successor list will fail is p^r. Increasing r makes the system more robust. By tuning this parameter, any degree of robustness with good reliability and fault resiliency may be achieved.

The following applications are examples of how Chord could be used:

- Cooperative mirroring or Cooperative File System (CFS) [22], in which multiple providers of content cooperate to store and serve each others' data. Spreading the total load evenly over all participant hosts lowers the total cost of the system, since each participant needs to provide capacity only for the average load, not for the peak load. There are two layers in CFS. The DHash (Distributed Hash) layer performs block fetches for the peer, distributes the blocks among the servers, and maintains cached and replicated copies. The Chord layer distributed lookup system is used to locate the servers responsible for a block.
- Chord-based DNS [23] provides a lookup service, with host names as keys and IP addresses (and other host information) as values. Chord could provide a DNS-like service by hashing each host name to a key [20]. Chord-based DNS would require no special servers, while ordinary DNS systems rely on a set of special root servers. DNS also requires manual management of the routing information (DNS records) that allows clients to navigate the name server hierarchy; Chord automatically maintains the correctness of the analogous routing information. DNS only works well when host names are hierarchically structured to reflect administrative boundaries; Chord imposes no naming structure. DNS is specialized to the task of finding named hosts or services, while Chord can also be used to find data object values that are not tied to particular machines.

C. Tapestry

Sharing similar properties as Pastry, Tapestry [7] employs decentralized randomness to achieve both load distribution and routing locality. The difference between Pastry and Tapestry is the handling of network locality and data object replication, and this difference will be more apparent, as described in Pasty section. Tapestry's architecture uses variant of the Plaxton *et al*. [1] distributed search technique, with additional mechanisms to provide availability, scalability, and adaptation in the presence of failures and attacks. Plaxton *et al*. proposes a distributed data structure, known as the Plaxton mesh, optimized to support a network overlay for locating named data objects which are connected to one root peer. On the other hand, Tapestry uses multiple roots for each data object to avoid single point of failure. In the Plaxton mesh, peers can take on the roles of servers (where data objects are stored), routers (forward messages), and clients (entity of requests). It uses

local routing maps at each peer, to incrementally route overlay messages to the destination ID digit by digit, for instance, $***7 \Rightarrow **97 \Rightarrow *297 \Rightarrow 3297$, where '$*$' is the wildcard, similar to the longest prefix routing in the CIDR IP address allocation architecture [24]. The resolution of digits from right to left or left to right is arbitrary. A peer's local routing map has multiple levels, where each of them represents a matching the suffix up to a digit position in the ID space. The n^{th} peer that a message reaches, shares a suffix of at least length n with the destination ID. To locate the next router, the $(n+1)^{th}$ level map is examined to locate the entry matching the value of the next digit in the destination ID. This routing method guarantees that any existing unique peer in the system can be located within at most $log_B N$ logical hops, in a system with N peers using NodeIDs of base B. Since the peer's local routing map assumes that the preceding digits all match the current peer's suffix, the peer needs only to keep a small constant size (B) entry at each route level, yielding a routing map of fixed constant size: $(entries/map)$ x $no.of maps = B.log_B N$.

The lookup and routing mechanisms of Tapestry is similar to Plaxton, which are based on matching the suffix in NodeID as described above. Routing maps are organized into routing levels, where each level contains entries that point to a set of peers closest in distance that matches the suffix for that level. Also, each peer holds a list of pointers to peers referred to as neighbors. Tapestry stores the locations of all data object replicas to increase semantic flexibility and allowing application level to choose from a set of data object replicas based on some selection criteria, such as date. Each data object may include an optional application-specific metric in addition to a distance metric; e.g. OceanStore [17] global storage architecture finds the closest cached document replica which satisfies the closest distance metric. These queries deviate from the simple *"find first"* semantics, and Tapestry will route the message to the closest k distinct data objects.

Tapestry handles the problem of a single point of failure due to a single data object's root peer by assigning multiple roots to each object. Tapestry makes use of *surrogate routing* to select root peers incrementally, during the publishing process to insert location information into Tapestry. Surrogate routing provides a technique by which any identifier can be uniquely mapped to an existing peer in the network. A data object's root or surrogate peer is chosen as the peer which matches the data object's ID, X. This is unlikely to happen, given the sparse nature of the NodeID space. Nevertheless, Tapestry assumes peer X exists by attempting to route a message to it. A route to a non-existent identifier will encounter empty neighbor entries at various positions along the way. The goal is to select an existing link, which can act as an alternative to the desired link; i.e. the one associated with a digit X. Routing terminates when a map is reached where the only non-empty routing entry belongs to the current peer. That peer is then designated as the surrogate root for the data object. While surrogate routing may take additional hops to reach a root if compared with Plaxton algorithm, the additional number of hops is small. Thus, surrogate routing in Tapestry has minimal routing overhead relative to the static global Plaxton algorithm.

Tapestry addresses the issue of fault adaptation and maintains cached content for fault recovery by relying on TCP timeouts and UDP periodic heartbeats packets, to detect link, server failures during normal operations, and rerouting through its neighbors. During fault operation, each entry in the neighbor map maintains two backup neighbors in addition to the closest/primary neighbor. On a testbed of 100 machines with 1000 peers simulations, the results in [103] shows that the good routing rates and maintenance bandwidths during instantaneous failures and continuing churn.

A variety of different applications have been designed and implemented on Tapestry. Tapestry is self-organizing, fault-tolerant, resilient under load, and it is a fundamental component of the OceanStore system [17], [25]. The OceanStore is a global-scale, highly available storage utility deployed on the PlanetLab [26] testbed. OceanStore servers use Tapestry to disseminate encoded file blocks efficiently, and clients can quickly locate and retrieve nearby file blocks by their ID, despite server and network failures. Other Tapestry applications include the Bayeux [27] — an efficient self organizing application-level multicast system and SpamWatch [28] — a decentralized spam-filtering system that uses a similarity search engine implemented on Tapestry.

D. Pastry

Pastry [4], like Tapestry, makes use of Plaxton-like prefix routing, to build a self-organizing decentralized overlay network, where each peer routes client requests and interacts with local instances of one or more applications. Each peer in Pastry is assigned a 128-bit peer identifier (NodeID). The NodeID is used to give a peer's position in a circular NodeID space, which ranges from 0 to $2^{128}-1$. The NodeID is assigned randomly when a peer joins the system and it is assumed to be generated such that the resulting set of NodeIDs is uniformly distributed in the 128-bit space. For a network of N peers, Pastry routes to the numerically closest peer to a given key in less than $log_B N$ steps under normal operation (where $B = 2^b$ is a configuration parameter with typical value of $b = 4$). The NodeIDs and keys are considered a sequence of digits with base B. Pastry routes messages to the peer whose NodeID is numerically closest to the given key. A peer normally forwards the message to a peer whose NodeIDs shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current peer NodeID.

As shown in Figure 5, each Pastry peer maintains a routing table, a neighborhood set, and a leaf set. A peer routing table is designed with $log_B N$ rows, where each row holds $B-1$ number of entries. The $B-1$ number of entries at row n of the routing table each refer to a peer whose NodeID shares the current peer's NodeID in the first n digits, but whose $(n+1)^{th}$ digit has one of the $B-1$ possible values other than the $(n+1)^{th}$ digit in the current peer's NodeID. Each entry in the routing table contains the IP address of peers whose NodeID have the appropriate prefix, and it is chosen according to close proximity metric. The choice of b involves a trade-off between the size of the populated portion of the routing table [$approx.(log_B N)$ x $(B-1)$ entries] and maximum number of hops required to route between any pair of peers $(log_B N)$.

Routing Table of a Pastry peer with NodeID 37A0x, b = 4, digits are in hexadecimal, x is an arbitrary suffix

0x	1x	2x	**3x**	4x	...	Dx	Ex	Fx
30x	31x	32x	...	**37x**	38x	...	3Ex	3Fx
370x	371x	372x	...	**37Ax**	37Bx	...	37Ex	37Fx
37A0x	37A1x	37A2x	...	37ABx	37ACx	37ADx	37AEx	37AFx

Example: Routing State of a Pastry peer with NodeID 37A0F1, b = 4, L=16, M=32

NodeID 37A0F1			
Leaf Set (Smaller)			
37A001	37A011	37A022	37A033
37A044	37A055	37A066	37A077
Leaf Set (Larger)			
37A0F2	37A0F4	37A0F6	37A0F8
37A0FA	37A0FB	37A0FC	37A0FE
Neighborhood Set			
1A223B	1B3467	245AD0	2670AB
3612AB	37890A	390AF0	3912CD
46710A	477810	4881AB	490CDE
279DE0	290A0B	510A0C	5213EA
11345B	122167	16228A	19902D
221145	267221	28989C	199ABC

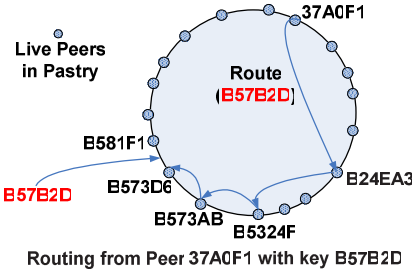

Fig. 5. Pastry peer's routing table, leaf set and neighbor set. An example of routing path for a Pastry peer.

With a value of $b = 4$ and 10^6 peers, a routing table contains on average 75 entries and the expected number of routing hops is 5. The neighborhood set, M, contains the NodeIDs and IP addresses of the $|M|$ peers that are closest in proximity to the local peer. The network proximity that Pastry uses is based on a scalar proximity metric such as the number of IP routing geographic distance. The leaf set L is the set of peers with $|L|/2$ numerically closest larger NodeIDs and $|L|/2$ peers with numerically smaller NodeIDs, in relation to the current peer's NodeID. Typical values for $|L|$ and $|M|$ are B or 2 x B. Even with concurrent peers failure, eventual delivery is guaranteed with good reliability and fault resiliency, unless $|L|/2$ peers with adjacent NodeIDs fail simultaneously, ($|L|$ is a configuration parameter with a typical value of 16 or 32).

When a new peer (NodeID is X) joins the network, it needs to initialize its state table and inform other peers of its presence. This joining peer needs to know the address of a contact peer in the network. A small list of contact peers, based on a proximity metric (e.g. the RTT to each peer) to provide better performance, could be provided as a service in the network, and the joining peer could select at random one of the peers to be its contact peer. So, this new peer knows initially about a nearby Pastry peer A, according to a proximity metric, from the list of contact peers. Peer X asks A to route a special *join* message with the key equal to X. Pastry routes the *join* message to the existing peer Z whose NodeID is numerically closest to X. In response to receiving the *join* request, peers A, Z and all peers encountered on the path from A to Z send their state tables to X. Finally, X informs any peers that need to be aware of its arrival. This ensures that X initializes its state with appropriate values and that the state in all other affected peers is updated. As peer A is assumed to be topologically close to the new peer X, A's neighborhood set initialize X's neighborhood set. Considering the general case, where NodeIDs of A and X share no common prefix, let A_i denote peer A's row of the routing table at level i. A_0 contains appropriate values for X_0, since the entries in row 0 of the routing table are independent of a peer's NodeID. Other levels of A's routing table are of no use to X, since A's and X's NodeIDs share no common prefix. Appropriate values for X_1 can be taken from B_1, when B is the first peer along the route path from A to Z. The entries in B_1 and X_1 share the same prefix, because X and B have the same first digit in their NodeID. Finally, X transmits a copy of its resulting state to each of the peers found in its neighborhood set, leaf set and routing table. These peers then update their own state based on the information received.

A Pastry peer is considered failed when its immediate neighbors in the NodeID space can no longer communicate with the peer. To replace a failed peer in the leaf set, its neighbor in the NodeID space contacts the live peer with the largest index on the side of the failed peer, and requests its leaf table. For example, if L_i failed for $|L|/2 < i < 0$, it requests the leaf set from $L - |L|/2$. Let the received leaf set be L', which overlaps the current peer's leaf set L, and it contains peers with nearby NodeIDs not present in L. The appropriate one is then chosen to insert into L, verifying that the peer is actually still alive by contacting it. To repair the failed routing table entry $R_{d(levell)}$, a peer contacts first the peer referred to by another entry $R_{i(levell)}, i \neq d$ of the same row, and asks for that peer's entry for R_d. If none of the entries in row l have a pointer to a live peer with appropriate prefix, the peer contacts an entry $R_{i(levell+1)}, i \neq d$, thereby casting a wider coverage. The neighborhood set is not used in the routing of messages, but it is still kept fresh/update because the set plays an important role in exchanging information about nearby peers. Therefore, a peer contacts each member of the neighborhood set periodically to see if it is still alive. If the peer is not responding, the peer asks other members for their neighborhood sets and checks for the closest proximity of each of the newly discovered peers and updates its own neighborhood set. Pastry is being used in the implementation of an application-level multicast, called SplitStream [29]. Instead of relying on a multicast infrastructure in the network which is not widely available, the participating peers route and distribute multicast message using only unicast network services. SplitStream allows a cooperative environment where peers contribute resources in exchange for using the service. The key idea is to split the content into k stripes and to multicast each stripe using a separate tree. Peers join as many trees as there are stripes they wish to receive and they

specify an upper bound on the number of stripes that they are willing to forward. The challenge is to construct this forest of multicast trees such that an interior peer in one tree is a leaf peer in all the remaining trees and the bandwidth constraints specified by the peers are satisfied. This ensures that forwarding load can be spread across all participating peers. For example, if all peers wish to receive k stripes and they are willing to forward k stripes, SplitStream will construct a forest such that the forwarding load is evenly balanced across all peers while achieving low delay and link stress across the network.

Scribe [30], [31] is a scalable application-level multicast infrastructure that supports a large number of groups with large number of members per group. Scribe is built on top of Pastry which is used to create and manage groups and to build efficient multicast trees for dissemination of messages to each group. Scribe builds a multicast tree formed by joining Pastry routes from each group member to a rendezvous point associated with a group. Membership maintenance and message dissemination in Scribe leverages the robustness, self-organization, locality and reliability properties of Pastry.

Squirrel [32] uses Pastry as its data object location service, to identify and route to peers that cache copies of a requested data object. It facilitates mutual sharing of web data objects among client peers, and enables the peers to export their local caches to other peers in the network, thus creating a large shared virtual web cache. Each peer then performs both web browsing and web caching, without the need for expensive and dedicated hardware for centralized web caching. Squirrel faces a new challenge whereby peers in a decentralized cache incur the overhead of having to serve each other requests, and this extra load must be kept low.

PAST [33], [34] is a large scale P2P persistent storage utility, based on Pastry. The PAST system is composed of peers connected to the Internet where each peer is capable of initiating and routing client requests to insert or retrieve files. Peers may also contribute storage to the system. A storage system like PAST is attractive because it exploits the multitude and diversity of peers in the Internet to achieve strong persistence and high availability. This eradicates the need for physical transport of storage media to protect lookup and archival data, and the need for explicit mirroring to ensure high availability and throughput for shared data. A global storage utility also facilitates the sharing of storage and bandwidth, thus permitting a group of peers to jointly store or publish content that would exceed the capacity or bandwidth of any individual peer.

Pastiche [35] is a simple and inexpensive backup system that exploits excess disk capacity to perform P2P backup with no administrative costs. The cost and inconvenience of backup are unavoidable, and often prohibitive. Small-scale solutions require significant administrative efforts. Large-scale solutions require aggregation of substantial demand to justify the capital costs of a large, centralized repository. Pastiche builds on three architecture: Pastry which provides the scalable P2P network with self-administered routing and peer location; Content-based indexing [36], [37], which provides flexible discovery of redundant data for similar files; and Convergent encryption [18] allows hosts to use the same encrypted representation for common data without sharing keys.

E. Kademlia

The Kademlia [14] P2P decentralized overlay network takes the basic approach of assigning each peer a NodeID in the 160-bit key space, and key,value pairs are stored on peers with IDs *close* to the key. A NodeID-based routing algorithm will be used to locate peers near a destination key. One of the key architecture of Kademlia is the use of a novel XOR metric for distance between points in the key space. XOR is symmetric and it allows peers to receive lookup queries from precisely the same distribution of peers contained in their routing tables. Kademlia can send a query to any peer within an interval, allowing it to select routes based on latency or send parallel asynchronous queries. It uses a single routing algorithm throughout the process to locate peers near a particular ID.

Every message being transmitted by a peer includes its peer ID, permitting the recipient to record the sender peer's existence. Data keys are also 160-bit identifiers. To locate {key,value} pairs, Kademlia relies on the notion of distance between two identifiers. Given two 160-bit identifiers, a and b, it defines the distance between them as their bitwise exclusive OR (XOR, interpreted as $d(a,b) = a \oplus b = d(b,a)$ for all a, b), and this is a non-Euclidean metric. Thus, $d(a,b) = 0, d(a,b) > 0 (if a \neq b)$, and for all a, b: $d(a,b) = d(b,a)$. XOR also offers the triangle inequality property: $d(a,b) + d(b,c) \geq d(a,c)$, since $d(a,c) = d(a,b) \oplus d(b,c)$ and $(a+b \geq a \oplus b)$ for all $a, b = 0$. Similarly to Chord's clockwise circle metric, XOR is unidirectional. For any given point x and distance $d > 0$, there is exactly one point y such that $d(x,y) = d$. The unidirectional approach makes sure that all lookups for the same key converge along the same path, regardless of the originating peer. Hence, caching {key,value} pairs along the lookup path alleviates hot spots.

The peer in the network stores a list of {IP address, UDP port, NodeID} triples for peers of distance between 2^i and 2^{i+1} from itself. These lists are called k-buckets. Each k-bucket is kept sorted by last time seen; i.e. least recently accessed peer at the head, most-recently accessed at the tail. The Kademlia routing protocol consists of:

- PING probes a peer to check if it is active.
- STORE instructs a peer to store a {key,value} pair for later retrieval.
- FIND_NODE takes a 160-bit ID, and returns {IP address,UDP port,NodeID} triples for the k peers it knows that are closest to the target ID.
- FIND_VALUE is similar to FIND_NODE, it returns {IP address,UDP port,NodeID} triples, except for the case when a peer received a STORE for the key, it just return the stored value.

Importantly, Kademlia's peer must locate the k closest peers to some given NodeID. This lookup initiator starts by picking X peers from its closest non-empty k-bucket, and then sends parallel asynchronous FIND_NODE to the X peers it has chosen. If FIND_NODE fails to return a peer that is any closer,

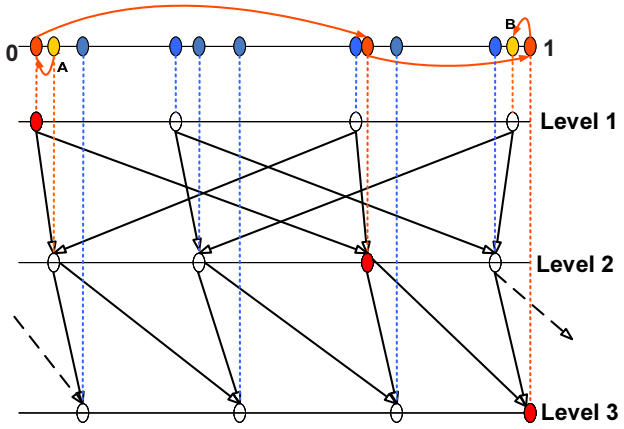

Fig. 6. A Simplified Viceroy network. For simplicity, the up link, ring and level-ring links are not shown.

than the closest peers already seen, the initiator resends the FIND_NODE to all of the k closest peers it has not already queried. It can route for lower latency because it has the flexibility of choosing any one of k peers to forward a request. To find a {key,value} pair, a peer starts by performing a FIND_VALUE lookup to find the k peers with IDs closest to the key. To join the network, a peer n must have contact to an already participating peer m. Peer n inserts peer m into the appropriate k-bucket, and then performs a peer lookup for its own peer ID. Peer n refreshes all k-buckets far away than its closest neighbor, and during this refresh, peer n populates its own k-buckets and inserts itself into other peers' k-buckets, if needed.

F. Viceroy

The Viceroy [15] P2P decentralized overlay network is designed to handle the discovery and location of data and resources in a dynamic butterfly fashion. Viceroy employs consistent hashing [20], to distribute data so that it is balanced across the set of servers and resilient to servers joining and leaving the network. It utilizes the DHT to manage the distribution of data among a changing set of servers and allowing peers to contact any server in the network to locate any stored resource by name. In addition to this, Viceroy maintains an architecture that is an approximation to a butterfly network [38], as shown in Figure 6 (adapted from diagram in [15]), and uses links between successors and predecessors - ideas that were based on Kleingberg [39] and Barrière *et al*. [40] — on the ring (a key is mapped to its successor on the ring) for short distances. Its diameter of the overlay is better than CAN and its degree is better than Chord, Tapestry and Pastry.

When N peers are operational, one of $logN$ levels is selected with near equal probability. Level l peer's two edges are connected to peers at level $l+1$. A down-right edge is added to a long-range contact at level $l+1$ at a distance about $\frac{1}{2^l}$ away, and a down-left edge at a close distance on the ring to the level $l+1$. The up edge to a nearby peer at level $l-1$ is included if $l > 1$. Then, level-ring links are added to the next and previous peers of the same level l. Routing is done by climbing using up connections to a level $l-1$ peer. Then proceeds down the levels of the tree using the down links, and moving from level l to level $l+1$. It follows either the edge to the nearby down link or the further down link, depending on distance $> \frac{1}{2^l}$. This is recursively continues until a peer is reached with no down links, and it is in the vicinity of the target peer. So, a vicinity lookup is performed using the ring and level-ring links. For reliability and fault resiliency, when a peer leaves the overlay network, it hands over its key pairs to a successor from the ring pointers and notifies other peers to find a replacement. It is formalized and proved [15] that the routing process requires only $O(logN)$, where N is the number of peers in the network.

III. DISCUSSION ON STRUCTURED P2P OVERLAY NETWORK

The algorithm of Plaxton was originally devised to route web queries to nearby caches, and this influenced the design of Pastry, Tapestry and Chord. The method of Plaxton has logarithmic expected join/leave complexity. Plaxton ensures that queries never travel further in network distance than the peer where the key is stored. However, Plaxton has several disadvantages: it requires global knowledge to construct the overlay; an object's root peer is the single point of failure; no insertion or deletion of peers; no avoidance to hotspots congestion. Pastry and Tapestry schemes relied on DHT to provide the substrate for semantic-free and data-centric references, through the assignment of a semantic-free NodeID, such as a 160-bit key, and performed efficient request routing between lookup peers using an efficient and dynamic routing infrastructure, whereby peers leave and join. Overlays that perform query routing in DHT-based systems have strong theoretical foundations, guaranteeing that a key can be found if it exists and they do not capture the relationships between the object name and its content. However, DHT-based systems have a few problems in terms of data object lookup latency:

1) For each overlay hop, peers route a message to the next intermediate peer that can be located very far away with regard to physical topology of the underlying IP network. This can result in high network delay and unnecessary long-distance network traffics, from a deterministic short overlay path of $O(logN)$, (where N is the number of peers).
2) DHT-based systems assume that all peers equally participate in hosting published data objects or their location information. This would lead to a bottleneck at low-capacity peers.

Pastry and Tapestry routing algorithms are a randomized approximation of a hypercube and routing towards an object is done by matching longer addresses suffixes, until either the object's root peer or another peer with a *nearby* copy is found. Rhea *et al*. [41] makes use of FreePastry implementation to discover that most lookups fail to complete when there is excessive churn. They claimed that short-lived peers leave the overlay with lookups that have not yet timed out. They outlined design issues pertaining to DHT-based performance under churn: lookup timeouts, reactive versus

periodic recovery of peers; and the choice of nearby neighbors. Since the reactive recovery will increase traffic to congested links, they make use of periodic recovery and for lookup, they suggested for an exponential weighted moving average of each neighbor's response time instead of alternative fixed timeout. They discovered that selection of nearby neighbors, required global sampling which is more effective than simply sampling neighbor's neighbors. However, Castro *et al*. [42] uses the MSPastry implementation to show that it can cope with high churn rates by achieving shorter routing paths and a lesser maintenance overhead. Pastry exploits network locality to reduce routing delays by measuring the delay Round-Trip-Time (RTT) to a small number of peers when building the routing tables. For each routing table entry, it chooses one of the closest peers in the network topology whose NodeID satisfies the constraints for that entry. The average IP delay of each Pastry hop increases exponentially until it reaches the average delay between two peers in the network. Chord's routing protocol is similar to Pastry's location algorithm in PAST. However, Pastry is a prefix-based routing protocol and differs in other details from Chord.

Chord maps keys and peers to an identifier ring and guarantees that queries make a logarithmic number hops and that keys are well balanced. It uses *Consistent Hashing* to minimize disruption of keys when peers leave and join the overlay network. Consistent Hashing ensures that the total number of caches responsible for a particular object is limited and when these caches changed, the minimum number of object references will move to maintain load balancing. Since the Chord lookup service presents a solution where each peer maintains a logarithmic number of long-range links, it gives a logarithmic join/leave updates. In Chord, the network is maintained appropriately by a background maintenance process, i.e. a periodic stabilization procedure that updates predecessor and successor pointers to cater for newly joined peers. Liben-Nowell *et al*. [43] asks the question of how often the stabilization procedure need to run to determine the success of Chord's lookups and determining the optimum involves the measurements of peers' behavior. Stoica *et al*. [6] demonstrates that the advantage of recursive lookups over iterative lookups, but future improvement work is proposed to improve resiliency to network partitions using a small set of known peers, and to reduce the amount of messages in lookups by increasing the size of each step around the ring with a larger fingers in each peer. Alima *et al*. [44] proposes an *correction-on-use* mechanism in their Distributed K-ary Search (DKS), which is similar to Chord, to reduce the communication costs incurred by Chord's stabilization procedure. The mechanism makes correction to the expired routing entries by piggybacking lookups and insertions.

The work on CAN has a constant degree network for routing lookup requests. It organizes the overlay peers into a d-dimensional Cartesian coordinate space, with each peer takes the ownership of a specific hyper-rectangular shape in the space. The key motivation of CAN design is based on the argument that Plaxton-based schemes would not perform well under churn, given peer departures and arrivals would affect a logarithmic number of peers. It maintains a routing table with its adjacent immediate neighbors. Peers joining the CAN cause the peer owning the region of space to split, giving half to the new peer and retaining half. Peers leaving the CAN will pass its NodeID, neighbors' NodeID, IP addresses and its {key,value} pairs to a takeover peer. CAN has a number of tunable parameters to improve routing performance: Dimensionality of the hypercube; Network-aware routing by choosing the neighbor closest to the destination in CAN space; multiple peers in a zone, allowing CAN to deliver messages to anyone of the peers in the zone in an anycast manner; uniform partitioning allowed by comparing the volume of a region with the volumes of neighboring regions when a peer joins; landmark-based placement causes peers, at join time, to probe a set of well known landmark hosts, estimating each of their network distances. There are open research questions on CAN's resiliency, load balancing, locality and latency/hopcount costs.

The Kademlia's XOR topology-based routing resembles very much the first phase in the routing algorithms of Pastry, Tapestry and Plaxton. For these three algorithms, there is a need for an additional algorithmic structure for discovering the target peer within the peers that share the same prefix but differ in the next b-bit digit. It was argued in [14] that Pastry and Tapestry algorithms require secondary routing tables of size $O(2^b)$ in addition to the main tables of size $O(2^b log_{2^b} N)$, which increases the cost of bootstrapping and maintenance. Kademlia resolves in their distinctive ways by the use of XOR metric for the *distance* between 160-bit NodeID and each peer maintains a list of contact peers, which longer-lived peers are given preference on this list. Kademlia can easily be optimized with a base other than 2, by configuring the bucket table so that it approaches the target b bits per hop. This needs having one bucket for each range of peers at a distance $[j.2^{160-(i+1)b}, (j+1).2^{160-(i+1)b}]$, for each $0 < j < 2^b$ and $0 \leq i < \frac{160}{b}$. This expects no more than $(2^b - 1)(log_{2^b} N)$ buckets.

The Viceroy overlay network (butterfly) presents an efficient network construction proved formally in [15], maintains constant degree networks in a dynamic environment, similar to CAN. Viceroy has logarithmic diameter, similar to Chord, Pastry and Tapestry. Viceroy's diameter is proven to be better than CAN and its degree is better than Chord, Pastry and Tapestry. Its routing achieved in $O(logN)$ hops (where N is the number of peers) and with nearly optimal congestion. Peers joining and leaving the system induce $O(logN)$ hops and require only $O(1)$ peers to change their states. Li *et al*. [45] describes in their paper that limited degree may increase the risk of network partition or limitations in the use of local neighbors. However, its advantage is the constant-degree overlay properties. Kaashoek *et al*. [46] highlights about its fault-tolerant blind spots and its complexity.

Further work were done by Viceroy's authors with proposal of a two-tier, locality-aware DHT [47] which gives lower degree properties in each lower-tier peer, and the bounded-degree P2P overlay using de Bruijn graph [48]. Since de Bruijn graphs give very short average routing distances and high resilience to peer failure, they are well suited for structured P2P overlay networks. The P2P overlays discussed above are

'greedy', and for a given degree, the algorithms are suboptimal because the routing distance is longer. There are increasing de Bruijn P2P overlay proposals [46], [49]–[52]. The de Bruijn graph of degree k (k can be varied) could achieve an asymptotically optimum diameter (maximum hopcounts between any two peers in the graph) of $log_k N$, where N is the total number of peers in the system. Given $O(logN)$ neighbors in each peer, the de Bruijn graphs' hop count is $O(\frac{logN}{loglogN})$. A good comparison study done by Loguinov *et al*. [50] where they use example of Chord, CAN and de Bruijn to study routing performance and resilience of P2P overlay networks, including graph expansion and clustering properties. They confirmed that de Bruijn graphs for a given degree k, offer the best diameter and average distance between all pairs of peers (this determines the expected response time in number of hops), optimal resilience (k-peer connectivity), large bisection width (bisection width of a graph provides tight upper bounds on the achievable capacity of the graph), and good node (peer) expansion that guarantees little overlap between parallel paths to any destination peer (if there is a peer failure, very few alternative paths to a destination peer are affected).

P2P DHT-based overlay systems are susceptible to security breach from malicious peers' attacks. One simple attack on DHT-based overlay system is that the malicious peer to return wrong data objects to the lookup queries. The authenticity of the data objects can be handled by using cryptographic techniques through some cost-effective public keys and/or content hashes to securely link together different pieces of data objects. Such techniques can neither prevent undesirable data objects from polluting the search results, nor preventing denial of attacks. Malicious peers may still be able to corrupt, deny access or response to lookup queries of replicas of a data object, and impersonate so that replicas may be stored on illegitimate peers.

Sit *et al*. [53] provides a very clear description of security considerations that involve the adversaries which are peers in the DHT overlay lookup system that do not follow the protocol correctly: malicious peers are able to eavesdrop the communication between other nodes; malicious peer can only receive data objects addressed to its IP address, and thus, IP address can be a weak form of peer identity; malicious peers can collude together giving believable false information. They presented a taxonomy of possible attacks involving routing deficiencies due to corrupted lookup routing and updates; vulnerability to partitioning and virtualization into incorrect networks when new peer join and contact malicious peer; lookup and storage attacks; inconsistent behaviors of peers; denial of service attacks preventing access by overloading victim's network connection; and unsolicited responses to a lookup query. Defenses design principles can be classified as defining verifiable system invariants for lookup queries, NodeID assignment, peers selection in routing, cross checking using random queries, and avoid single point of responsibility. Castro *et al*. [54] relates the problems of secure routing for structured P2P overlay networks, in terms of the possibilities that a small number of peers could compromise the overlay system, if peers are malicious and conspire with each other (this is also termed as Eclipse attack [55]). They presented a design and analysis of techniques for secure peer joining, routing table maintenance, and robust message forwarding in the presence of malicious peers in Structured P2P overlays. The technique can tolerate up to 25% of malicious peers while providing good performance when the number of compromised peers is small. However, this defense restricts the flexibility necessary to implement optimizations such as proximity neighbor selection and only works in Structured P2P overlay networks. So, Singh *et al*. [55] proposes a defense that prevents Eclipse attacks for both Structured and Unstructured P2P overlay networks, by bounding degree of overlay peers, i.e. the in-degree of overlay peers is likely to be higher than the average in-degree of legitimate peers and legitimate peers choose their neighbors from a subset of overlay peers whose in-degree is below a threshold. Having done the in-degree bounding, it is still possible for the attacker to consume the in-degree of legitimate peers and prevent other legitimate peers from referencing to them, therefore, bounding the out-degree is necessary so that legitimate peers choose neighbors from the subset of overlay peers whose in-degree and out-degree are below some threshold. An auditing scheme is also introduced to prevent incorrect information of the in-degree and out-degree.

Another good survey on security issues in P2P is from Wallach [56], describe that secured routing primitives: assigning NodeIDs, maintaining routing tables, and forwarding of messages securely. He also suggested looking at distributed auditing the sharing of disk space resources in P2P overlay network as a barter economy, and the mechanism to implement such an economy. The work on *BarterRoam* [57] sheds light on a formal computational approach that is applicable to P2P overlay systems towards exchanging resources so that higher level functionality, such as incentive-compatible economic mechanisms can be layered at the higher layers. Formal game theoretical approach and models [58]–[60] could be constructed to analyze equilibrium of user strategies to implement incentives for cooperation. The ability to overcome free-rider problems in P2P overlay networks will definitely improve the system's reliability and its value

Sybil attack termed by Douceur [61] described the situation whereby there are a large number of potentially malicious peers in the system and without a central authority to certify peers' identities. It becomes very difficult to trust the claimed identity. Dingledine *et al*. [62] proposes puzzles schemes, including the use of micro-cash, which allow peers to build up reputations. Although this proposal provides a degree of accountability, this still allows a resourceful attacker to launch attacks. Many P2P computational models of trust and reputation systems have emerged to assess trustworthiness behavior through feedback and interaction mechanisms. The basic assumption of these computational trust and reputation models is that the peers engage in bilateral interactions and evaluations done on a globally agreed scale. However, most of such trust and reputation systems suffer from two problems, as highlighted by Despotovic *et al*. [63]: extensive implementation overhead and vague trust related model semantics. The causes lie in the aggregation of the feedback about all peers in

the overlay network in order to assess the trustworthiness of a single peer, and also the anti-intuitive feedback aggregation strategies resulting in outputs that are difficult to interpret. They proposed a simple probabilistic estimation technique with maximum likelihood estimation suffice to reduce these problems when the feedback aggregation strategy is employed.

Lastly, since each of these basic Structured P2P DHT-based systems defines different methods in the higher level DHT abstractions to map keys to peers and other Structured P2P application-specific systems such as cooperative storage, content distribution and messaging; there is recent effort [12] in defining basic common API abstractions for the common services they provide which they called key-based routing API (KBR) at lower tier of the abstractions. At higher tier, more abstractions can be built upon this basic KBR. In addition to DHT abstraction which provides the same functionality as the hash table in Structured P2P DHT-based systems by mapping between keys and objects, the group anycast and multicast (CAST) (provides scalable group communication and coordination) and decentralized object location and routing (DOLR) (provides a decentralized directory service) are also defined. However, Karp *et al*. [13] points out that the above mentioned *bundled library* model where the applications read the local DHT state and receive *upcalls* from the DHT, requires the codes for the same set of applications to be available at all DHT hosts. This prevents the sharing of a single DHT deployment by multiple applications and generates maintenance traffic from running the DHT on its underlying infrastructure. Thus, they proposed OpenHash with ReDiR, a distributed rendezvous service model that requires only a *put()/get()* interfaces and shares a common DHT routing platform. The authors argued that this open DHT service will spur more development of DHT-based applications without the burden of deploying and maintaining a DHT.

Table I summarizes the characteristics of *Structured* P2P overlay networks which have been discussed in section II.

IV. UNSTRUCTURED P2P OVERLAY NETWORKS

In this category, the overlay networks organize peers in a random graph in flat or hierarchical manners (e.g. Super-Peers layer) and use flooding or random walks or expanding-ring Time-To-Live (TTL) search, etc. on the graph to query content stored by overlay peers. Each peer visited will evaluate the query locally on its own content, and will support complex queries. This is inefficient because queries for content that are not widely replicated must be sent to a large fraction of peers and there is no coupling between topology and data items' location. In this section, we shall survey and compare some of the more seminal Unstructured P2P overlay networks: Freenet [64], Gnutella [9], FastTrack [65]/KaZaA [66], BitTorrent [67], Overnet/eDonkey2000 [68], [69].

A. Freenet

Freenet is an adaptive P2P network of peers that make query to store and retrieve data items, which are identified by location-independent keys. This is an example of *loosely Structured* decentralized P2P network with placement of files based on anonymity. Each peer maintains a dynamic routing table, which contains addresses of other peers and the data keys that they are holding. The key features of Freenet are the ability of maintaining locally a set of files in accordance to the maximum disk space allocated by the network operator, and to provide security mechanisms against malicious peers. The basic model is that requests for keys are passed along from peer to peer through a chain of proxy requests in which each peer makes a local decision about the location to send the request next, similar to the Internet Protocol (IP) routing. Freenet also enables users to share unused disk space, thus allowing a logical extension to their own local storage devices.

The basic architecture consists of data items being identified by binary file keys obtained by applying the 160-bit SHA-1 hash function [70]. The simplest type of file key is the Keyword-Signed Key (KSK), which is derived from a short descriptive text string chosen by the user, e.g. `/music/Britney.Spears`. The descriptive text string is used as the input to deterministically generate a public/private key pair, and the public half is then hashed to yield the data file key. The private half of the asymmetric key pair is used to sign the data file, thus, providing a minimal integrity check that a retrieved data file matches its data file key. The data file is also encrypted using the descriptive string itself as a key, so as to perform an explicit lookup protocol to access the contents of their data-stores.

However, nothing prevents two users from independently choosing the same descriptive string for different files. These problems are addressed by the Signed-Subspace Key (SSK), which enables personal namespaces. The public namespace key and the descriptive string are hashed independently, XOR'ed together and hashed to yield the data file key. For retrieval, the user publishes the descriptive string together with the user subspace's public key. Storing data requires the private key, so that only the owner of a subspace can add files to it, and owners have the ability to manage their own namespaces. The third type of key in FreeNet is the Content-Hash Key (CHK), which is used for updating and splitting of contents. This key is derived from hashing the contents of the corresponding file, which gives every file a pseudo-unique data file key. Data files are also encrypted by a randomly generated encryption key. For retrieval, the user publishes the content-hash key itself together with the decryption key. The decryption key is never stored with the data file but is only published with the data file key, so as to provide a measure of cover for operators. The CHK can also be used for splitting data files into multiple parts in order to optimize storage and bandwidth resources. This is done by inserting each part separately under a CHK and creating an indirect file or multiple levels of indirect files to point to the individual parts. The routing algorithm for storing and retrieving data is designed to adaptively adjust routes over time and to provide efficient performance while using local knowledge, since peers only have knowledge of their immediate neighbors. Thus, the routing performance is good for popular content. Each request is given a Hops-To-Live (HTL) limit, similar to the IP Time-To-Live (TTL) which is decremented at each peer to prevent infinite chains. Each request is also assigned a pseudo-unique random identifier,

TABLE I
A COMPARISON OF VARIOUS *Structured* P2P OVERLAY NETWORK SCHEMES

Algorithm Taxonomy	Structured P2P Overlay Network Comparisons					
	CAN	Chord	Tapestry	Pastry	Kademlia	Viceroy
Decentralization	DHT functionality on Internet-like scale					
Architecture	Multi-dimensional ID coordinate space.	Uni-directional and Circular NodeID space.	Plaxton-style global mesh network.	Plaxton-style global mesh network.	XOR metric for distance between points in the key space.	Butterfly network with connected ring of predecessor and successor links, data managed by servers.
Lookup Protocol	key,value pairs to map a point P in the coordinate space using uniform hash function.	Matching Key and NodeID.	Matching suffix in NodeID.	Matching Key and prefix in NodeID.	Matching Key and Node-ID based routing.	Routing through levels of tree until a peer is reached with no downlinks; vicinity search performed using ring and level-ring links.
System Parameters	N-number of peers in network d-number of dimensions.	N-number of peers in network.	N-number of peers in network B-base of the chosen peer identifier.	N-number of peers in network b-number of bits ($B = 2^b$) used for the base of the chosen identifier.	N-number of peers in network b-number of bits (($B = 2^b$) of NodeID.	N-number of peers in network.
Routing Performance	$O(d.N^{\frac{1}{d}})$	$O(logN)$	$O(log_BN)$	$O(log_BN)$	$O(log_BN)+c$ where c = small constant	$O(logN)$
Routing State	$2d$	$logN$	log_BN	$Blog_BN + Blog_BN$	$Blog_BN + B$	$logN$
Peers Join/Leave	$2d$	$(logN)^2$	log_BN	log_BN	log_BN+c where c = small constant	$logN$
Security	Low level. Suffers from man-in-middle and Trojan attacks.					
Reliability/Fault Resiliency	Failure of peers will not cause network-wide failure. Multiple peers responsible for each data item. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data on multiple consecutive peers. On failures, application retries.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers. Keep track of multiple paths to each peer.	Failure of peers will not cause network-wide failure. Replicate data across multiple peers.	Failure of peers will not cause network-wide failure. Load incurred by lookups routing evenly distributed among participating lookup servers.

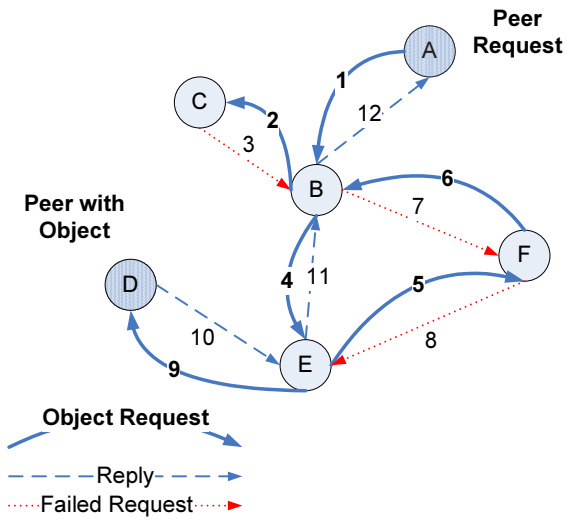

Fig. 7. A typical request sequence in Freenet.

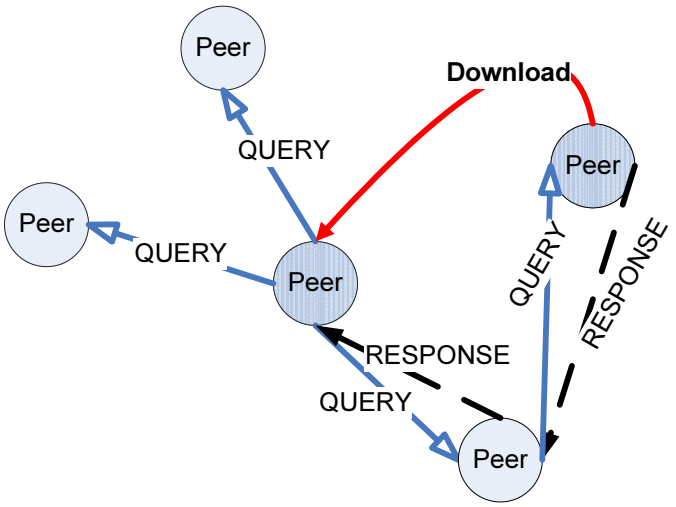

Fig. 8. Gnutella utilizes a decentralized architecture document location and retrieval.

so that peers can avoid loops by rejecting requests they have seen before. If this happens, the preceding peer chooses a different peer to forward to. This process continues until the request either is satisfied or has exceeded its HTL limit. The success or failure signal (message) is returned back up the chain to the sending peer. Joining the network will rely on discovering the address of one or more of the existing peers through out-of-band means, and no peer is privileged over any other peer, so no hierarchy or centralized point of failure can exist. This intuitive resilience and decentralization enhances the performance and scalability, thus, giving a constant routing state while peers joins and leaves the overlay.

In addition, as described in [64], Freenet uses its datastore to increase system performance. When an object is returned (forwarded) after a successful retrieval (insertion), the peer caches the object in its datastore, and passes the object to the upstream (downstream) requester which then creates a new entry in its routing table associating the object source with the requested key. So, when a new object arrives from either a new insert or a successful request, this would cause the datastore to exceed the designated size and Least Recently Used (LRU) objects are ejected in order until there is space. LRU policy is also applied to the routing table entries when the table is full.

Figure 7 depicts a typical sequence of request messages. The user initiates a data request at peer A, which forwards the request to peer B, and then forwards it to peer C. Peer C is unable to contact any other peer and returns a backtracking *failed request* message to peer B. Peer B tries its second choice, peer E, which forwards the request to peer F, which then delivers it to peer B. Peer B detects the loop and returns a backtracking failure message. Peer F is unable to contact any other peer and backtracks one step further back to peer E. Peer E forwards the request to its second choice, peer D, which has the data. The data is returned from peer D, via peers E, B and A. The data is cached in peers E, B and A, therefore, it creates a routing short-cut for the next similar queries. This example shows the overlay suffers from security problems such as man-in-middle and Trojan attacks, and the failure of peers will not cause network-wide failure, because of its lack of centralized structure. This gives good reliability and fault resiliency.

B. Gnutella

Gnutella (pronounced *newtella*) is a decentralized protocol for distributed search on a flat topology of peers (servents). Gnutella is widely used and there has been a large amount of work on improving Gnutella [71]–[73]. Although the Gnutella protocol supports a traditional client/centralized server search paradigm, Gnutella's distinction is its peer-to-peer, decentralized model for document location and retrieval, as shown in Figure 8. In this model, every peer is a server or client. This system is neither a centralized directory nor does it possess any precise control over the network topology or file placement. The network is formed by peers joining the network following some loose rules. The resultant topology has certain properties, but the placement of data items is not based on any knowledge of the topology, as in the *Structured* P2P designs. To locate a data item, a peer queries its neighbors, and the most typical query method is flooding. The lookup query protocol is flooded to all neighbors within a certain radius. Such design is extremely resilient to peers entering and leaving the system. However, the current search mechanisms are not scalable and generating unexpected loads on the network.

The so-called Gnutella servents (peers) perform tasks normally associated with both clients and servers. They provide client-side interfaces through which users can issue queries and view search results, while at the same time they also accept queries from other servents, check for matches against their local data set, and respond with applicable results. These peers are responsible for managing the background traffic that spreads the information used to maintain network integrity. Due to the distributed nature, a network of servents that implement the Gnutella protocol is highly fault-tolerant, as operation of the network will not be interrupted if a subset of servents goes offline.

To join the system, a new servent (peer) initially connects

to one of several known hosts that are almost always available, e.g. list of peers available from `http://gnutellahosts.com`. Once connected to the network, peers send messages to interact with each other. These messages are broadcasted, (i.e. sent to all peers with which the sender has open TCP connections), or simply back-propagated, (i.e. sent on a specific connection on the reverse of the path taken by an initial, broadcasted message). First, each message has a randomly generated identifier. Second, each peer keeps a short memory of the recently routed messages, used to prevent re-broadcasting and to implement back-propagation. Third, messages are flagged with TTL and "hops passed" fields. The messages that are allowed in the network are:

- Group Membership (PING and PONG) Messages. A peer joining the network initiates a broadcasted PING message to announce its presence. The PING message is then forwarded to its neighbors and initiates a back-propagated PONG message, which contains information about the peer such as the IP address, number and size of the data items.
- Search (QUERY and QUERY RESPONSE) Messages. QUERY contains a user specified search string that each receiving peer matches against locally stored file names and it is broadcasted. QUERY RESPONSE are back-propagated replies to QUERY messages and include information necessary to download a file.
- File Transfer (GET and PUSH) Messages. File downloads are done directly between two peers using these types of messages.

Therefore, to become a member of the network, a servent (peer) has to open one or many connections with other peers that are already in the network. With such a dynamic network environment, to cope with the unreliability after joining the network, a peer periodically PINGs its neighbors to discover other participating peers. Peers decide where to connect in the network based only on local information. Thus, the entire application-level network has servents as its peers and open TCP connections as its links, forming a dynamic, self-organizing network of independent entities.

The latest versions of Gnutella uses the notion of super-peers or ultra-peers [11](peers with better bandwidth connectivity), to help improve the routing performance of the network. However, it is still limited by the flooding mechanism used for communications across ultra-peers. Moreover, the ultra-peer approach makes a binary decision about a peer's capacity (ultra-peer or not) and to our knowledge, it has no mechanism to dynamically adapt the ultra-peer-client topologies as the system evolves. Ultra-peers perform query processing on behalf of their leaf peers. When a peer joins the network as a leaf, it selects a number of ultra-peers, and then it publishes its file list to those ultra-peers. A query for a leaf peer is sent to an ultra-peer which floods the query to its ultra-peer neighbors up to a limited number of hops. Dynamic querying [74] is a search technique whereby queries that return fewer results are re-flooded deeper into the network.

Saroiu *et al*. [75] examines the bandwidth, latency, availability and file sharing patterns of the peers in Gnutella and Napster, and highlighted the existence of significant heterogeneity in both systems. Krishnamurthy *et al*. [76] proposes a cluster-based architecture for P2P systems (CAP), which uses a network-aware clustering technique (based on a central clustering server) to group peers into clusters. Each cluster has one or more delegate peers that act as directory servers for objects stored at peers within the same cluster. Chawathe *et al*. [73] proposes a model called Gia, by modifying Gnutella's algorithms to include flow control, dynamic topology adaptation, one-hop replication and careful attention to peer heterogeneity. The simulation results suggest that these modifications provide three to five orders of magnitude improvement in the total capacity of the system while retaining significant robustness to failures. Thus, by making a few simple changes to Gnutella's search operations would result in dramatic improvements in its scalability.

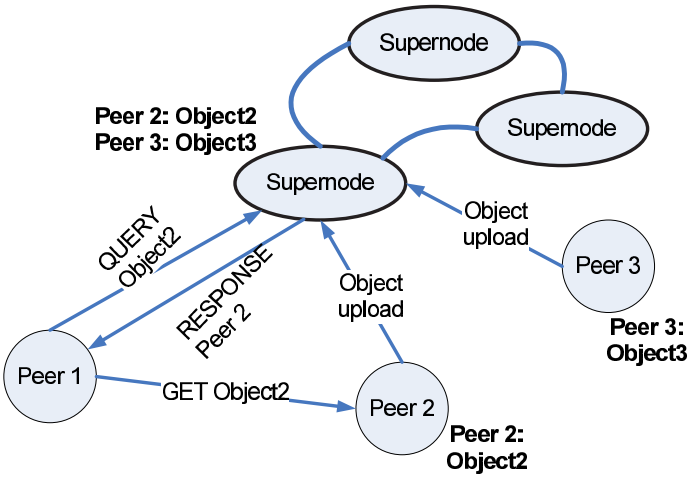

Fig. 9. FastTrack peers connect to Superpeers whereby the search is routed through the Superpeers and downloads are done from the peer peers.

C. FastTrack/KaZaA

FastTrack [65] P2P is a decentralized file-sharing system that supports meta-data searching. Peers form a structured overlay of super-peers architecture to make search more efficient, as shown in Figure 9. Super-peers are peers with high bandwidth, disk space and processing power, and have volunteered to get elected to facilitate search by caching the meta-data. The ordinary peers transmit the meta-data of the data files they are sharing to the super-peers. All the queries are also forwarded to the super-peer. Then, Gnutella-typed broadcast based search is performed in a highly pruned overlay network of super-peers. The P2P system can exist, without any super-peer but this result in worse query latency. However, this approach still consumes bandwidth so as to maintain the index at the super-peers on behalf of the peers that are connected. The super-peers still use a broadcast protocol for search and the lookup queries is routed to peers and super-peers that have no relevant information to the query. Both KaZaA [66] and Crokster [77] are both FastTrack applications.

As mentioned, KaZaA is based on the proprietary FastTrack

protocol which uses specially designated super-peers that have higher bandwidth connectivity. Pointers to each peer's data are stored on an associated super-peer, and all queries are routed to the super-peers. Although this approach seems to offer better scaling properties than Gnutella, its design has not been analyzed. There have been proposals to incorporate this approach into the Gnutella network [11]. The KaZaA peer to peer file sharing network client supports a similar behavior, allowing powerful peers to opt-out of network support roles that consume CPU and bandwidth.

KaZaA file transfer traffic consists of unencrypted HTTP transfers; all transfers include KaZaA-specific HTTP headers (e.g., *X-KaZaA-IP*). These headers make it simple to distinguish between KaZaA activity and other HTTP activity. The KaZaA application has an auto-update feature; meaning, a running instance of KaZaA will periodically check for updated versions of itself. If it is found, it downloads the new executable over the KaZaA network.

A power-law topology, commonly found in many practical networks such as WWW [78], [79], has the property that a small proportion of peers have a high out-degree (i.e., have many connections to other peers), while the vast majority of peers have a low out-degree, (i.e., have connections to few peers). Formally, the frequency f_d of peers with out-degree d exhibits a power-law relationship of the form $f_d \propto d^a, a < 0$. This is the Zipf property with Zipf distributions looking linear when plotted on a log-log scale. Faloutsos *et al*. [80] has found that Internet routing topologies follow this power-law relationship with $a \approx -2$. However, Gummadi *et al*. [75], [81] observes that the KaZaA measured popularity of the file-sharing workload does not follow a Zipf distribution. The popularity of the most requested objects (mostly large, immutable video and audio objects) is limited, since clients typically fetch objects at most once, unlike the Web. Thus the popularity of KaZaA's objects tends to be short-lived, and popular objects tend to be recently born. There is also significant locality in the KaZaA workload, which means that substantial opportunity for caching to reduce wide-area bandwidth consumption.

D. BitTorrent

BitTorrent [67] is a centralized P2P system that uses a central location to manage users' downloads. This file distribution network uses *tit-for-tat* (peer responds with the same action that its other collaborating peer performed previously) as a method of seeking. The protocol is designed to discourage free-riders, by having the peers choose other peers from which the data has been received. Peers with high upload speed will probably also be able to download with a high speed, thus achieving high bandwidth utilization. The download speed of a peer will be reduced if the upload speed has been limited. This will also ensure that content will be spread among peers to improve reliability.

The architecture consists of a central location which is a tracker that is connected to when you download a *.torrent* file, which contains information about the file, its length, name, and hashing information, and URL of a tracker, as illustrated in Figure 10. The tracker keeps track of all the peers who have the file (both partially and completely) and lookup peers to connect with one another for downloading and uploading. The trackers use a simple protocol layered on top of HTTP in which a downloader sends information about the file it is downloading and the port number. The tracker responds with a random list of contact information about the peers which are downloading the same file. Downloaders then use this information to connect to each other. A downloader which has the complete file, known as a *seed*, must be started, and send out at least one complete copy of the original file.

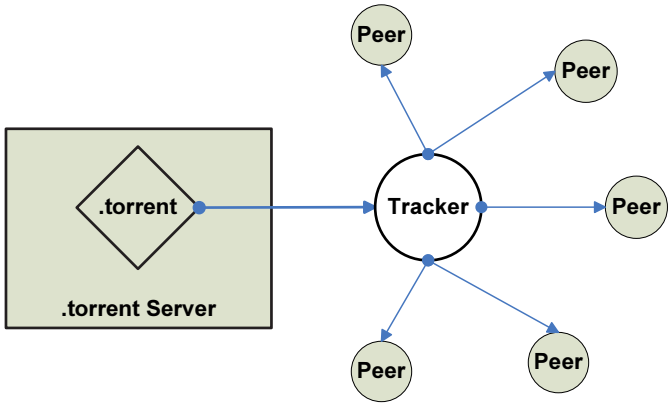

Fig. 10. BitTorrent architecture consists of centralized Tracker and .torrent file.

BitTorrent cuts files into pieces of fixed size (256 Kbytes) so as to track the content of each peer. Each downloader peer announces to all of its peers the pieces it has, and uses SHA1 to hash all the pieces that are included in the *.torrent* file. When a peer finishes downloading a piece and checks that the hash matches, it announces that it has that piece to all of its peers. This is to verify data integrity. Peer connections are symmetrical. Messages sent in both directions look the same, and data can flow in either direction. When data is being transmitted, downloader peers keep several requests (for pieces of data) queued up at once in order to get good TCP performance. This is known as *pipelining*. Requests which cannot be written out to the TCP buffer immediately are queued up in memory rather than kept in an application-level network buffer, so they can all be thrown out when a choke happens.

Choking is a temporary refusal to upload; downloading can still happen and the connection does not need to be renegotiated when choking stops. Choking is done for several reasons. TCP congestion control behaves very poorly when sending over many connections at once. Additionally, choking lets each peer use a *tit-for-tat*-like algorithm to ensure that they get a consistent download rate. There are several criteria that a good choking algorithm should meet. It should cap the number of simultaneous uploads for good TCP performance. It should avoid choking and unchoking quickly, known as *fibrillation*. It should reciprocate service access to peers who let it download.

Finally, it should try out unused connections once in a while to find out if they might be better than the currently used ones, known as optimistic unchoking.

The currently deployed BitTorrent choking algorithm avoids fibrillation by only changing the peer that is choked once every ten seconds. It does reciprocation and the number of uploads are capped by unchoking the four peers with the best download rates and have interest in. Peers which have a better upload rate but are not interested get unchoked and if they become interested, the worst uploader gets choked. If a downloader has a complete file, it uses its upload rate rather than its download rate to decide which to unchoke. For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of its upload rate. If this peer is interested, it counts as one of the four allowed downloaders. Peers which are optimistically unchoked rotate every 30 seconds.

E. Overnet/eDonkey2000

Overnet/eDonkey [68], [69] is a hybrid two-layer P2P information storage network composed of client and server, which are used to publish and retrieve small pieces of data by creating a file-sharing network. This architecture provides features such as concurrent download of a file from multiple peers, detection of file corruption using hashing, partial sharing of files during downloading and expressive querying methods for file search. To join the network, the peer (client) needs to know the IP address and port of another peer (server) in the network. It then bootstraps from the other peer. The clients connect to a server and register the object files that they are sharing by providing the meta-data describing the object files. After registration, the clients can either search by querying the meta-data or request a particular file through its unique network identifier, thus providing guarantee service to locate popular objects. Servers provide the locations of object files when requested by clients, so that clients can download the files directly from the indicated locations.

V. DISCUSSION ON UNSTRUCTURED P2P OVERLAY NETWORK

The Unstructured P2P centralized overlay model was first popularized by Napster. This model requires some managed infrastructure (the directory server) and show some scalability limits. A flooding requests model for decentralized P2P overlay systems such as Gnutella, whereby each peer keeps a user-driven neighbor table to locate data objects are quite effective to locate popular data objects, thanks to the power-law property of user-driven characteristics. However, it can lead to excessive network bandwidth consumption, and remote or unpopular data objects may not be found due to the limit of lookup horizon, typically imposed by TTL.

The argument is that DHT-based systems while more efficient at many tasks and have strong theoretical fundamentals to guarantee a key to be found if it exists, they are not well suited for mass-market file sharing. They do not capture the semantic object relationships between its name and its content or metadata. In particular, DHT-based ability to find exceedingly rare objects is not required in a mass-market file sharing environment, and their ability to efficiently implement keyword search is still not proven. In addition, they use precise placement algorithms and specific routing protocols to make searching efficient. However, these Structured P2P overlay systems have not been widely deployed, and their ability to handle unreliable peers has not been tested. Thus, in the research community, efforts are being made in improving the lookup properties of Unstructured P2P overlays to include flow control, dynamic geometric topology adaptation, one-hop replication, peer heterogeneity, etc.

The Freenet, like Chord, does not assign responsibility for data to specific peers, and its lookups take the form of searches for cached copies. This prevents it from guaranteeing retrieval of existing data or from providing low bounds on retrieval costs. But Freenet provides anonymity and it introduces a novel indexing scheme where files are identified by *content-hash* keys and by secured *signed-subspace* keys to ensure that only one object owner can write to a file and anyone can read it. P2P overlay designs using DHTs share similar characteristics as Freenet — an exact query yields an exact response. This is not surprising since Freenet uses a hash function to generate keys. Recent research in [82] shows that that changing Freenet's routing table cache replacement scheme from LRU to enforcing clustering in the key space can significantly improve performance. This idea is based on the intuition from the small-world models [39] and theoretical results by Kleinberg [39].

Version 0.6 of the Gnutella protocol [9], [10] adopted the concept of ultra-peers which are high capacity peers that act as proxies for lower capacity peers. One of the main enhancements is the Query Routing Protocol (QRP), which allow the leaf peers to forward index of object name keywords to its ultra-peers [83]. This allows the ultra-peers to have their leaves to receive lookup queries when they have a match, and subsequently, it reduces the lookup query traffic at the leaves. A shortcoming of QRP is that the lookup query propagation is independent of the popularity of the objects. The Dynamic Query Protocol [84] addressed this by letting the leaf peers to send single queries to high-degree ultra-peers which adjust the lookup queries' TTL bounds in accordance to its number of received lookup query results. The Gnutella UDP Extension for Scalable Searches (GUESS) [85] also aimed to reduce the number of lookup queries by repeatedly queries single ultra-peers with a TTL of 1, to limit load on each lookup query.

As described in the Gnutella section in this paper, Chawathe *et al*. [73] improves the Gnutella design using their Gia system, by incorporating adaptation algorithm so that peers are attached to high-degree peers and providing a receiver-based token flow control for sending lookup queries to neighbors. Instead of flooding, they make use of random walk search algorithm and also the system keep pointers to objects in neighboring peers. However, in [85], they proposed that Unstructured P2P overlay like Gnutella can be built on top of Structured P2P overlay to help reduce the lookup queries overhead and overlay maintenance traffic. They used the *collapse point* lookup query rate (define as the per node query rate at which the successful query rate falls below 90%) and the average hopcounts prior to collapse. However, the comparison

was done in static network scenario with the older Gnutella and not the enhanced Gnutella version.

BitTorrent, a second generation P2P overlay system, achieves higher level of robustness and resource utilization based on its incentives cooperation technique for file distribution. The longest and most comprehensive measurement study of BitTorrent P2P system [86] provides a more insight by comparing a detailed measurement study of BitTorrent with other popular P2P file-sharing systems, such as FastTrack/KaZaA, Gnutella, Overnet/eDonkey, and DirectConnect, based on five characteristics:

1) Popularity - Total number of users participating over a certain period of time.
2) Availability - System availability depending on contributed resources.
3) Download Performance - Contrast between size of data and the time required for download.
4) Content Lifetime - Time period when data is injected into the system till no peers is willing to share the data anymore.
5) Pollution Level - Fraction of corrupted content spread throughout the system.

FastTrack/KaZaA has the largest file sharing community, with Overnet/eDonkey and BitTorrent gaining popularity. The popularity of BitTorrent system is influenced by the availability of the central components in terms of its number of downloads and technical faults in the system. Availability has a significant influence on popularity. FastTrack/KaZaA being more advanced in architecture has good availability because of its Super-Peers that allow the network to scale very well, by creating indexing. Gnutella and Overnet/eDonkey provide full and partial distribution of the responsibility for shared files respectively. The availability of content in BitTorrent is unpredictable and vulnerable to potential failures, due to its lack of decentralization. BitTorrent is well-suited for download performance due to its advanced download distribution protocol. Overnet/eDonkey takes an opposite approach by offering powerful searching capabilities and queue-based scheduling of downloads, which can take longer waiting times. The lack of archive functionality in BitTorrent results in relatively short content lifetimes. FastTrack/KaZaA which uses directory-level sharing policy allows data files to be located as long as the peer holding the data file stays connected. FastTrack/KaZaA system does not limit the number of fake files in the overlay but it allow user to identify correct files based on hash-code verification. BitTorrent prevents fake files from floating in the system. The arising use of firewalls and NATs are growing problems for P2P systems because of the effect of reducing the download speed. This proposal [87] tries to solve the firewall problems by designing a hybrid CDN structure with a P2P based streaming protocol in the access network based on an empirical study of BitTorrent, which found that the need for additional freeloader protection and the potential negative effect of firewall on download speeds. A fluid model for BitTorrent-like networks was proposed [88] to capture the behavior of the system when the arrival rate is small, and to study the steady-state network performance. The study also provided expressions for the various parameters, such as average number of sees, the average number of downloaders, and the average downloading time, and proved that Nash equilibrium exists for each peer chooses its uploading bandwidth to be equal to the actual uploading bandwidth.

It is also interesting to note that most of these Unstructured P2P network (such as KaZaA and Gnutella), are not pure power-law networks with Zipf distribution properties; for example, analysis in [89] shows that Gnutella networks have topologies that are power-law random graphs, and later measurement shows that there are too few peers with a low number of connectivity. This may be attributed to the behaviors of the users of these P2P networks. Research on power-law networks [78], [80], [90], [91] shows that networks as diverse as the Internet, organize themselves so that most peers have few links while a small number of peers have a large number of links. The interesting paper by Adamic *et al*. [92] studies random-walk search strategies in power-law networks, and discovers that by changing walkers to seek out high degree peers, the search performance can be optimized greatly. Several search techniques for Unstructured P2P networks are discussed in [93]: iterative deepening, directed BFS and local indices. Networks with power-law organizational structure, display an unexpected degree of robustness [94], i.e. the ability of the peers to communicate unaffectedly even by extremely high failure rates. But these networks are prone to attacks.

Thus, Unstructured P2P networks reduce the network dependence on a small number of highly connected, easy to attack peers. Instead of using DHT as building blocks in distributed applications, SkipNet [95] is a new overlay based on Skip Lists that organizes peers and data primarily by their sorted string names, as Skip Lists do, rather than by hashes of those names. In this way, Skip Net supports useful locality properties as well as the usual DHT functionality. In addition, some recent research, e.g. Loo *et al*. [96] proposes the design of a hybrid model by using Structured DHT techniques to locate rare object items, and Unstructured flooding to locate highly replicated contents.

All of the security issues discussed in the Structured P2P overlay networks section applies to Unstructured P2P overlay networks. It is worthwhile to highlight the work from Bellovin [97], which it is difficult to limit Napster and Gnutella use via firewalls and how information can be leaked through search queries in the overlay network. The work highlighted concern over Gnutella's *push* feature, intended to work around firewalls, which might be useful for distributed denial of service attacks. Napster's centralized architecture might be more secure towards such attacks due to a centralized trusted server.

Table II summarizes the comparisons of *Unstructured* P2P networks. While by no means comprehensive, we believe that it captures the essence of the discussion and analysis done in the previous section IV.

VI. CONCLUDING REMARKS

This paper has presented various schemes in *Structured* and *Unstructured* P2P overlay networks that have been proposed

TABLE II
A COMPARISON OF VARIOUS *Unstructured* P2P OVERLAY NETWORK SCHEMES

Algorithm Taxonomy	Unstructured P2P Overlay Network Comparisons				
	Freenet	Gnutella	FastTrack/KaZaA	BitTorrent	Overnet/eDonkey2000
Decentralization	Loosely DHT functionality.	Topology is flat with equal peers.	No explicit central server. Peers are connected to their Super-Peers.	Centralized model with a Tracker keeping track of peers.	Hybrid two-layer network composed of clients and servers.
Architecture	Keywords and descriptive text strings to identify data objects.	Flat and Ad-Hoc network of servents (peers). Flooding request and peers download directly.	Two-level hierarchical network of Super-Peers and peers.	Peers request information from a central Tracker.	Servers provide the locations of files to requesting clients for download directly.
Lookup Protocol	Keys,Descriptive Text String search from peer to peer.	Query flooding.	Super-Peers.	Tracker.	Client-Server peers.
System Parameters	None	None	None	*.torrent* file.	None
Routing Performance	Guarantee to locate data using *Key* search until the requests exceeded the Hops-To-Live (HTL) limits.	No guarantee to locate data; Improvements made in adapting Ultrapeer-client topologies; Good performance for popular content.	Some degree of guarantee to locate data, since queries are routed to the Super-Peers which has a better scaling; Good performance for popular content.	Guarantee to locate data and guarantee performance for popular content.	Guarantee to locate data and guarantee performance for popular content.
Routing State	Constant	Constant	Constant	Constant but choking (temporary refusal to upload) may occur.	Constant
Peers Join/Leave	Constant	Constant	Constant	Constant	Constant with bootstrapping from other peer and connect to server to register files being shared.
Security	Low; Suffers from man-in-middle and Trojan attacks.	Low; Threats: flooding, malicious content, virus spreading, attack on queries, and denial of service attacks.	Low; Threats: flooding, malicious or fake content, viruses, etc. Spywares monitor the activities of peers in the background.	Moderate; Centralized Tracker manage file transfer and allows more control which makes it much harder faking IP addresses, port numbers, etc.	Moderate; Similar threats as the FastTrack and BitTorrent.
Reliability/Fault Resiliency	No hierarchy or central point of failure exists.	Degradation of the performance; Peer receive multiple copies of replies from peers that have the data; Requester peers can retry.	The ordinary peers are reassigned to other Super-Peers.	The Tracker keeps track of the peers and availability of the pieces of files; Avoid Choking by fibrillation by changing the peer that is choked once every then seconds.	Reconnecting to another server; Will not receive multiple replies from peers with available data.

by researchers. The P2P overlay network that is best suited depends on the application and its required functionalities and performance metrics for example, scalability, network routing performance, location service, file sharing, content distribution, and so on. Several of these schemes are being applied to the sharing of music, replication of electronic address books, multi-player games, provisioning of mobile, location or ad-hoc services, and the distribution of workloads of mirrored websites.

Finally, we close this survey with our thoughts on some directions for the future in P2P overlay networking research:

- The concerns of how well the P2P overlay networks' virtual topology maps to the physical network infrastructure, which has impact on the additional stress on the infrastructure, will undoubtedly, incur costs for the service providers. It would be useful to provide a quantitative evaluation on P2P overlay application and Internet topology matching and the scalability of P2P overlay applications by the efficient use of the underlying physical network resources.
- Having some sort of incentive model using economic and game theories, for P2P peers to collaborate is crucial to create an economy of equilibrium. When non-cooperative users benefit from free-riding on others' resources, the *tragedy of the commons* [98] is inevitable. Such incentives implementation in P2P overlay services would also provide a certain level of self-regulatory auditing and accounting behavior for resource sharing.
- Trust and reputation is also important for secured and trustworthy P2P overlay communications among the peers. For example, Kademlia developed a trust and secured architecture for routing and location service, by discouraging free-riding based on honesty, and routing away from the defective or malicious peer.
- Proximity in P2P overlay routing is an important factor in the routing decision for P2P overlay networks, which could improve global routing properties. There are on-going research in this area based on mapping the peers into coordinate-based space [99]–[107] and heuristic proximity routing optimizations [108]–[112]. Taking heterogeneity of the peers into account when delegating responsibility across peers overlay will improve the routing scalability. Future research would aim to reduce the stretch (ratio of overlay path to underlying network path) routing metric based on scalable and robust proximity calculations. This leads to improved P2P overlay operations performance globally. A mixed set of metrics which include delay, throughput, available bandwidth and packet loss would provide a more efficient global routing optimization.
- Application of P2P overlay networking models in mobile and ad-hoc wireless network. Because of their similar features, such as self-organizing, peer-to-peer routing, and resilient communications, application of P2P overlay approaches would allow mobile peers to have optimized flow control, load balancing mechanism, proximity-aware and QoS routing [113].

We see the future of P2P overlay networks inexorably linked to the take-up and subsequent commercial success of P2P overlay computing, personal area and ad-hoc networking, mobile location-based services, mirrored content delivery, and networked file-sharing. We may also conjecture that the prevalent problems of the Internet — controlling spam, maintaining directory services, and multicasting content have intuitive solutions from various P2P overlay network schemes. However, in order to move forward, the development community needs to understand the applicability of various schemes for *Structured* and *Unstructured* P2P overlay network models. This survey has been a modest attempt to address this need.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments.

REFERENCES

[1] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1997.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distribution: Evidence and implications," in *Proceedings of the IEEE INFOCOM*, 1999.

[3] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the ACM Symposium on Theory of Computing*, May 1997, pp. 654–663.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the Middleware*, 2001.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in *Processings of the ACM SIGCOMM*, 2001, pp. 161–172.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[7] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[8] Napster. [Online]. Available: http://www.napster.com/

[9] (2001) Gnutella development forum, the gnutella v0.6 protocol. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/

[10] (2002) Gnucleus, the gnutella web caching system. [Online]. Available: http://www.gnucleus.net/gwebcache/

[11] (2002) Gnutella ultrapeers. [Online]. Available: http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm/

[12] F. Dabek, B. Zhao, P. Druschel, and I. Stoica, "Towards a common api for structured peer-to-peer overlays," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, California, USA, February 20-21 2003.

[13] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker, "Spurring adoption of dhts with openhash, a public dht service," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, Berkeley, California, USA, February 26-27 2004.

[14] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Processings of the IPTPS*, Cambridge, MA, USA, February 2002, pp. 53–65.

[15] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in *Processings of the ACM PODC'02*, Monterey, CA, USA, July 2002, pp. 183–192.

[16] P. Francis, "Yoid: Extending the internet multicast architecture," unpublished, April 2000. http://www.aciri.org/yoid/docs/index.html.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *Processings of the ACM ASPLOS*, November 2002.

[18] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," in *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, June 2000, pp. 34–43.

[19] M. Waldman, A. D. Rubin, and L. F. Cranor, "Publius: a robust, tamper-evident, censorship-resistant, web publishing system," in *Processings of the Ninth USENIX Security Symposium*, Denver, CO, USA, 2000.

[20] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, May 1997, pp. 654–663.

[21] "Secure hash standard," NIST, U.S. Dept. of Commerce, National Technical Information Service FIPS 180-1, April 1995.

[22] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 202–215.

[23] R. Cox, A. Muthitacharoen, and R. Morris, "Serving dns using chord," in *Proceedings of the First International Workshop on Peer-to-Peer Systems*, March 2002.

[24] Y. Rekhter and T. Li. (1993) An architecture for ip address allocation with cidr. IETF Internet draft RFC1518. [Online]. Available: http://www.isi.edu/in-notes/rfc1518.txt/

[25] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global storage in oceanstore," in *IEEE Internet Computing*, 2001.

[26] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 59–64, 2003.

[27] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, 2001, pp. 11–20.

[28] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. D. Kubiatowicz, "Approximate object location and spam filtering on peer-to-peer systems," in *Processings of the Middleware*, June 2003.

[29] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, October 19-20 2003, pp. 298–313.

[30] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," in *Proceedings of the Third International Workshop on Networked Group Communications (NGC2001)*, London, UK, November 2001, pp. 30–43.

[31] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," in *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, October 2002.

[32] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC)*, Monterey, California, USA, July 21-24 2002.

[33] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," in *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*. Schloss Elmau, Germany: IEEECompSoc, May 2001.

[34] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, October 2001, pp. 188–201.

[35] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: making backup cheap and easy," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 285–298, 2002.

[36] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001, pp. 174–187.

[37] U. Manber, "Finding similar files in a large file system," in *Proceedings of the USENIX Winter 1994 Conference*, January 1994, pp. 1–10.

[38] H. J. Siegel, "Interconnection networks for simd machines," *Computer*, vol. 12, no. 6, pp. 57–65, 1979.

[39] J. Kleinberg, "The small-world phenomenon: an algorithm perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, 2000, pp. 163–170.

[40] L. Barriére, P. Fraigniaud, E. Kranakis, and D. Krizanc, "Efficient routing in networks with long range contacts," in *Proceedings of the 15th International Conference on Distributed Computing*, vol. 2180, 2001, pp. 270–284.

[41] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *Proceedings of the 2nd International Workshop on Peer-to-Peer (IPTPS'03)*, February 2003.

[42] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, Palazzo dei Congressi, Florence, Italy, June 28 - July 1 2004.

[43] D. Liben-Nowell, H. Balakrishnan, and D. Karger, "Analysis of the evolution of peer-to-peer systems," in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, Monterey, California, USA, 2002.

[44] L. Alima, S. El-Ansary, P. Brand, and S. Haridi, "Dks(n,k,f): a family of low communication, scalable and fault-tolerant infrastructures for p2p applications," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Monterey, California, USA, 2003, pp. 344–350.

[45] X. Li and C. Plaxton, "On name resolution in peer to peer networks," in *Proceedings of the 2nd ACM International workshop on principles of mobile computing*, Monterey, California, USA, 2002, pp. 82–89.

[46] F. Kaashoek and D. Karger, "Koorde: A simple degree-optimal hash table," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, CA, USA, February 20-21 2003.

[47] I. Abraham, D. Malkhi, and O. Dubzinski, "Land: Stretch (1+epsilon) locality aware networks for dhts," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, New Orleans, LA., USA, 2004.

[48] N. D. de Bruijn, "A combinatorial problem," *Koninklijke Netherlands: Academe Van Wetenschappen*, vol. 49, pp. 758–764, 1946.

[49] M. Naor and U. Wieder, "Novel architectures for p2p applications: the continuous-discrete approach," in *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2003)*, San Diego, California, USA, June 7-9 2003, pp. 50–59.

[50] D. Loguinov, A. Kumar, and S. Ganesh, "Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience," in *Proceedings of the ACM SIGCOMM*, Karlsruhe, Germany, August 25-29 2003, pp. 395–406.

[51] M. Naor and U. Wieder, "A simple fault tolerant distributed hash table," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, Berkeley, California, USA, February 20-21 2003.

[52] P. Fraigniaud and P. Gauron, "The content-addressable networks d2b," Laboratoire de Recherche en Informatique, Universit'e de Paris Sud, Tech. Rep. Technical Report 1349, Jan. 2003.

[53] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA.

[54] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 299–314, 2002.

[55] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against eclipse attacks on overlay networks," in *Proceedings of the SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[56] D. S. Wallach, "A survey of peer-to-peer security issues," in *Proceedings of the International Symposium on Software Security*, Tokyo, Japan, November 2002.

[57] E. K. Lua, A. Lin, J. Crowcroft, and V. Tan, "Barterroam: A novel mobile and wireless roaming settlement model," in *Proceedings of the QofIS*, 2004, pp. 348–357.

[58] C. Buragohain, D. Agrawal, and S. Suri, "A game-theoretic framework for incentives in p2p systems," in *Proceedings of the IEEE P2P 2003*, Linkoping, Sweden, September 1-3 2003.

[59] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," *Lecture Notes in Computer Science*, vol. 2232, pp. 75+, 2001.

[60] K. Lai, M. Feldman, J. Chuang, and I. Stoica, "Incentives for cooperation in peer-to-peer networks," in *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Linkoping, Sweden, June 2003.

[61] J. R. Douceur, "The sybil attack," in *Proceedings of the First International Workshop on Peer-to-Peer Systems*, March 7-8 2002, pp. 251–260.

[62] R. Dingledine, M. J. Freedman, and D. Molnar, "Accountability measures for peer-to-peer systems," in *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, D. Derickson, Ed. O'Reilly and Associates, November.

[63] Z. Despotovic and K. Aberer, "A probabilistic approach to predict peers' performance in p2p networks," in *Proceedings of the Eighth International Workshop on Cooperative Information Agents (CIA 2004)*, Erfurt, Germany, September 27-29 2004.

[64] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. (1999) Freenet: A distributed anonymous information storage and retrieval system. Freenet White Paper. [Online]. Available: http://freenetproject.org/freenet.pdf

[65] (2001) Fasttrack peer-to-peer technology company. [Online]. Available: http://www.fasttrack.nu/

[66] (2001) Kazaa media desktop. [Online]. Available: http://www.kazaa.com/

[67] (2003) Bittorrent. [Online]. Available: http://bitconjurer.org/BitTorrent/

[68] (2002) The overnet file-sharing network. [Online]. Available: http://www.overnet.com/

[69] (2000) Overnet/edonkey2000. [Online]. Available: http://www.edonkey2000.com/

[70] "Public key cryptography for the financial services industry - part 2: The secure hash algorithm (sha-1)," American National Standards Institute, Tech. Rep. American National Standard X9.30.2-1997, 1997.

[71] P. Ganesan, Q.Sun, and H. Garcia-Molina, "Yappers: A peer-to-peer lookup service over arbitrary topology," in *Proceedings of the IEEE Infocom 2003*, San Francisco, USA, March 30 - April 1 2003.

[72] Q. Lv, S. Ratnasamy, and S. Shenker, "Can heterogeneity make gnutella scalable?" in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, February 2002.

[73] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proceedings of the ACM SIGCOMM*, Karlsruhe, Germany, August 25-29 2003.

[74] Gnutella proposals for dynamic querying. [Online]. Available: http://www9.limewire.com/developer/dynamic_query.html/

[75] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, California, USA, January 2002.

[76] B. Krishnamurthy, J. Wang, and Y. Xie, "Early measurement of a cluster-based architecture for p2p systems," in *Proceedings of the ACM SIGCOOM Internet Measurement Workshop*, San Francisco, USA, November 2001.

[77] Grokster. [Online]. Available: http://www.grokster.com/

[78] A. Barabási, R. Albert, H. Jeong, and G. Bianconi, "Power-law distribution of the world wide web," *Science*, vol. 287, 2000.

[79] R. Albert, H. Jeong, and A. Barabási, "Diameter of the world wide web," *Nature*, vol. 401, pp. 130–131, 1999.

[80] M. Faloutsos, P.Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Processings of the SIGCOMM 1999*, 1999.

[81] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file sharing workload," in *Processings of the SOSP*, Bolton Landing, New York, USA, October 19-22 2003.

[82] A. Goel and R. Govindan, "Using the small-world model to improve freenet performance," *Computer Networks Journal*, vol. 46, no. 4, pp. 555–574, November 2004.

[83] A. Singla and C. Rohrs. Ultrapeers: Another step towards gnutella scalability. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Proposals/Working_Proposals/Ultrapeer/

[84] A. Fisk. Gnutella ultrapeer query protocol v0.1. [Online]. Available: http://groups.yahoo.com/group/the_gdf/files/Proposals/Working_Proposals/search/Dynamic_Querying/

[85] S. Daswani and A. Fisk. Gnutella udp extension for scalable searches (guess) v0.1. [Online]. Available: http://www.limewire.org/fisheye/viewrep/~raw,r=1.2/limecvs/core/guess_01.html

[86] J. A. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "A measurement study of the bittorrent peer-to-peer file sharing system," Delft University of Technology Parallel and Distributed Systems Report Series, Tech. Rep. Technical Report PDS-2004-007, 2004.

[87] K.-A. Skevik, V. Goebel, and T. Plagemann, "Analysis of bittorrent and its use for the design of a p2p based streaming protocol for a hybrid cdn," Delft University of Technology Parallel and Distributed Systems Report Series, Tech. Rep. Technical Report, June 2004.

[88] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proceedings of the ACM SIGCOMM*, Karlsruhe, Germany, August 30 - September 3 2004.

[89] M. A. Jovanovic, F. Annexstein, and K. Berman, "Scalability issues in large peer-to-peer networks - a case study of gnutella," Delft University of Technology Parallel and Distributed Systems Report Series, University of Cincinnati, Tech. Rep. Technical Report, 2001. [Online]. Available: http://www.ececs.uc.edu/~mjovanov/Research/paper.html

[90] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stat, A. Tomkins, and J. Wiener, "Graph structure in the web," in *Processings of the 9th Int. WWW Conference*, Amsterdam, May 15-19 2000.

[91] A. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 509, 1999.

[92] L. Adamic, R. Lukose, A. Puniyani, and B. Huberman, "Search in power-law networks," *Physical Review E*, vol. 64, 2001.

[93] B. Yang and H. Garcia-Molina, "Efficient search in peer-to-peer networks," in *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, July 2002.

[94] H. J. R. Albert and A. Barabási, "Attack and tolerance in complex networks," *Nature*, vol. 406, no. 378, 2000.

[95] N. J. A. Harvey, M. B. Jones, S. S. amd M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties," in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle, WA, USA, March 2003.

[96] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, California, USA, February 26-27 2004.

[97] S. Bellovin, "Security aspects of napster and gnutella," in *Proceedings of the 2001 Usenix Annual Technical Conference*, Boston, Massachusetts, USA, June 2001.

[98] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, pp. 1243–1248, 1968.

[99] E. K. Lua, J. Crowcroft, and M. Pias, "Highways: Proximity clustering for scalable peer-to-peer network," in *Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, August 25-28 2004, pp. 266–267.

[100] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical Internet Coordinates for Distance Estimation," in *24th IEEE International Conference on Distributed Computing Systems (ICDCS' 04)*, Tokyo, Japan, March 2004.

[101] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proceedings of the ACM SIGCOMM '04 Conference*, Portland, Oregon, August 2004.

[102] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," in *Proceedings of the IEEE INFOCOM '03 Conference*, San Francisco, California, USA, March 30 - April 3 2003.

[103] ——, "On the curvature of the internet and its usage for overlay construction and distance estimation," in *Proceedings of the IEEE INFOCOM '04 Conference*, Hong Kong, March 7-11 2004.

[104] T. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *IEEE INFOCOM' 02*, New York, USA, June 2002.

[105] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for scalable distributed location," in *2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[106] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *ACM SIGCOMM Internet Measurement Conference (IMC'03)*, Miami (FL), USA, October 2003.

[107] H. Lim, J. Hou, and C. Choi, "Constructing internet coordinate system based on delay measurement," in *ACM SIGCOMM Internet Measurement Conference (IMC'03)*, Miami (FL), USA, October 2003.

[108] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in peer-to-peer overlay networks," Microsoft Research Laboratory, Tech. Rep. Technical report MSR-TR-2002-82, 2002.

[109] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proceedings of the IEEE Infocom*, 2002.

[110] D. R. Karger and M.Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, May 2002.

[111] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," in *Proceedings of the IEEE Infocom*, 2002.

[112] M. Waldvogel and R. Rinaldi, "Efficient topology-aware overlay network," in *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, October 2002.

[113] J. Li, J. Jannotti, D. D. Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proceedings of the 6th ACM International Conference Mobile Computing and Networking*, Boston, Massachusetts, USA, August 2000, pp. 120–130.