A Methodology for the Design of Distributed Search in P2P Middleware

Jan Mischke[1] and Burkhard Stiller[2,1]

[1] Computer Engineering and Networks Laboratory TIK, Swiss Federal Institute of Technology (ETH Zurich)
Gloriastrasse 35, CH – 8092 Zürich, Switzerland
[2] Information Systems Laboratory IIS, University of Federal Armed Forces Munich
Werner-Heisenberg-Weg 39, D-85577 Neubiberg, Germany

E-Mail: [mischke|stiller]@tik.ee.ethz.ch

Abstract

Important research efforts are being conducted in the area of search, lookup, and routing, and are even increasing in the quest for P2P middleware that is both scalable and decentralized. To structure and classify current as well as to facilitate and give direction to future research, this methodology proposes a top-down two-dimensional design space. This design space has been developed for exhaustiveness so as to cover all possible design options, existing or yet to be conceived. A comprehensive survey of P2P search systems serves as a reference for the reader while at the same time validating the framework. An identification of areas in the design space not being covered by current systems leads to the design of a novel peer-to-peer-based keyword routing scheme. Finally, an evaluation of possible design options along the most important requirements will help guide system designers.

Keywords: *Distributed Systems, Middleware, Peer-to-Peer, Overlay Topology, Design Space, Keyword Search, Lookup, Semantic Routing*

1 Introduction

The difficulty of finding and retrieving or using networked resources, *i.e.*, content, services, or hardware, is increasing with the network size and degree of decentralization. While it was rather easy in times of mainframe computing with only few connected terminals, the move towards decentralized peer-to-peer (P2P) systems with millions of active nodes imposes huge challenges on distributed search and routing.

A P2P system is typically built upon existing network infrastructure providing end-to-end connectivity (Figure 1). P2P applications like filesharing, grid computing, or instant messaging require search middleware building on overlay networks to overcome the hurdles of decentralization and to search the network: *e.g.*, for files, computing resources, or users. Other P2P middleware functionality might be necessary, for instance for peer and content reputation information handling, and can build on the same or on separate overlay networks; the scope of this work, however, is restricted to search middleware.

Innumerable efforts have been started to build such middleware, to construct suitable overlay networks, and hence to design P2P search, lookup, and routing systems. However, a structure and delineation of these designs is yet missing as well as a comparative evaluation; attempts so far have been limited to a few randomly selected example systems [1], [2]. Furthermore, there is no statement as to what designs are viable at all and may not even have been looked into.

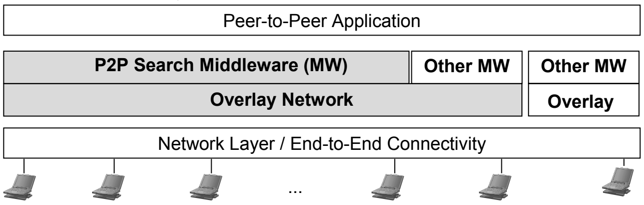
Peer-to-Peer Application
P2P Search Middleware (MW)
Other MW
Other MW
Overlay Network
Overlay
Network Layer / End-to-End Connectivity
...

Figure 1: P2P Networks: Systems View and Layering

Consider peer-to-peer file-sharing services as an example. Napster provided a central directory server to enable users to find content - it failed, mostly due to legal issues with its centralized architecture. Gnutella, in contrast, chose a decentralized approach based on flooding - but it can obviously not scale to the millions of nodes expected to join future P2P systems. Chord [3], CAN [4], Tapestry [5], Pastry [6], or AGILE [7] designed highly scalable combined lookup and routing systems - however, their highly structured approach makes them vulnerable to malicious users and makes keyword search a non-trivial task.

So several questions arise: What are the parallels or differences between these and further approaches that make them better or less well suited for one or the other application, and what are the trade-offs? Which systems have already been developed? And are there any fundamentally new approaches yet to be discovered and developed that may achieve significant performance leaps? These and further issues will be addressed in this paper.

While structured distributed hash tables like Chord are usually applied to combined lookup and routing, and unstructured Gnutella-like networks to search for keywords, it turned out to be important to separate the functionality certain P2P search middleware provides from the structural approach chosen. Section 2 identifies the *functional options* like name routing or semantic routing.

It is then shown that the number of different *structural approaches* is limited and the same for each functional option. A design space is defined with the goal to be complete for current and future systems (Section 3), based upon a classification into mutually exclusive and collectively exhaustive categories.

The framework is tested in Section 4 by mapping more than 30 existing approaches onto the two-dimensional design space and classification. At the same time, this section serves as a survey of P2P and distributed search systems. Finally, it allows the identification of blind spots in the current research landscape that can be filled by a novel approach for P2P search that is yet to be developed.

An evaluation of the design options and their trade-offs is given in Section 5, before Section 6 concludes.

2 Functionality of P2P Search Middleware

For a complete picture of functional options, search has been disaggregated into several steps. Figure 2 (top) shows the process from *keywords* over *names* and *addresses* to the *path to target node* hosting the desired resources.

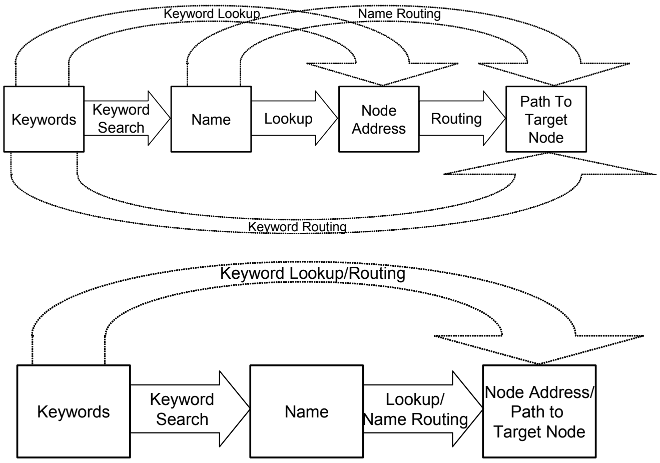

Figure 2: Search in Distributed (top) and P2P (bottom) Systems

Usually, a user wants to specify what he is looking for in terms of *keywords*. In the simplest case, keywords are just one or more terms appearing in the desired content or describing the desired resource. More sophisticated approaches apply content/resource meta information based on attribute-value pairs, *e.g.*, the resource description framework (RDF). *Keyword search* describes the functionality of mapping the resource meta information onto one, or, in the case of multiple matching resources, several unique *names* or identifiers in the network. Examples of such names are the Uniform Resource Locator, URL, or file names in a Unix file system. *Lookup* maps unique names onto *addresses* in the network. Addresses specify the network location of the node hosting the resource with a given name, *e.g.*, the IP address of the host. Finally, *routing* is the process of finding a *path* and moving queries to the *target node*.

Three short-cut mechanisms can help optimize search. *Name routing* combines the (distributed) lookup of the target node address with path identification and query forwarding to that node. *Keyword lookup* returns one or more addresses of nodes hosting resources with given keyword descriptions. Napster is the most prominent example. Finally, *keyword routing* directly routes towards a node hosting specified resources. Keyword routing is sometimes also called *semantic routing* or *content routing*.

For P2P, the process can be simplified as shown in Figure 2 (bottom). Since P2P systems apply application-level overlay networks, routing becomes a trivial task: knowing the target node address, the requestor simply creates a new virtual link to that address. Only few circumstances (like the anonymity requirement in Freenet) lead to a more difficult overlay routing approach, which is, however, an issue separate from search.

3 Structural Design Space and Classification

With the search process defined and disaggregated, it becomes obvious that searching requires a series of mappings, from the keyword space to the name space to the address space to the space of paths to nodes. The structural options in a distributed environment are the same for each mapping, and a classification is given in Figure 3.

A mapping can only be defined through a *computation* or a *table*. A (pre-defined) computation is difficult to achieve but some attempts have been made, usually involving hashing. More widely adopted are *tables* with entries for the desired search items, *e.g.*, a node address for each valid name. Mapping then comes down to finding the desired table entry and looking up the associated value. In a distributed environment, a table can either reside on a *central* entity like a search engine server, or be completely *replicated* on each node, or be *distributed* among the nodes.

Distributed tables are probably most challenging in that they require for each mapping to collaboratively find and contact the node that has the desired information or table entry. Two important aspects distinguish distributed table approaches: the *structure of the table*, *i.e.* the distribution of table entries to nodes, and the physical or overlay *topology* of the network. The distribution of table entries can either happen at random or according to a target table structure; the same applies for the distribution of links and, hence, the topology. Whether the table structure and topology are designed and aligned, or both random, or at least one of them designed but not aligned with the other, has substantial implications on search.

In a *random table structure and random topology,* it is natural that each node at least carries information about itself, *i.e.* its address, the names of its resources and content, and corresponding keyword descriptions. In addition to information on their own tables, nodes may have knowledge on the table entries of their *neighbors*, *i.e.* the nodes they directly know about and may contact for search, in an aggregated or non-aggregated form. The knowledge on neighboring table entries will in some cases be restricted to the direct neighbors, but can also involve *recursion*: An arbitrary node A not only learns about the table entries of its neighbors B_i, but also through B_i about B_i's neighbors C_{ij}, C_{ij}'s neighbors D_{ijk}, and so on. This way, nodes eventually know about most or even all keywords, names, or addresses in the direction of each neighbor in a usually aggregated way.

Rather than keeping explicit knowledge on neighboring table entries, nodes can exploit *implicit knowledge* when the table distribution and topology follow an *aligned structure* that every node knows. The most common approach is certainly the *classical hierarchy.* A root node informs about table areas represented by a number of second-level nodes. The second-level nodes, in turn, delegate to third-level nodes for sub-areas within their own area, and so on, until a request finally reaches the leaf node responsible for the desired entry. Particularly in the quest for scalable peer-to-peer search

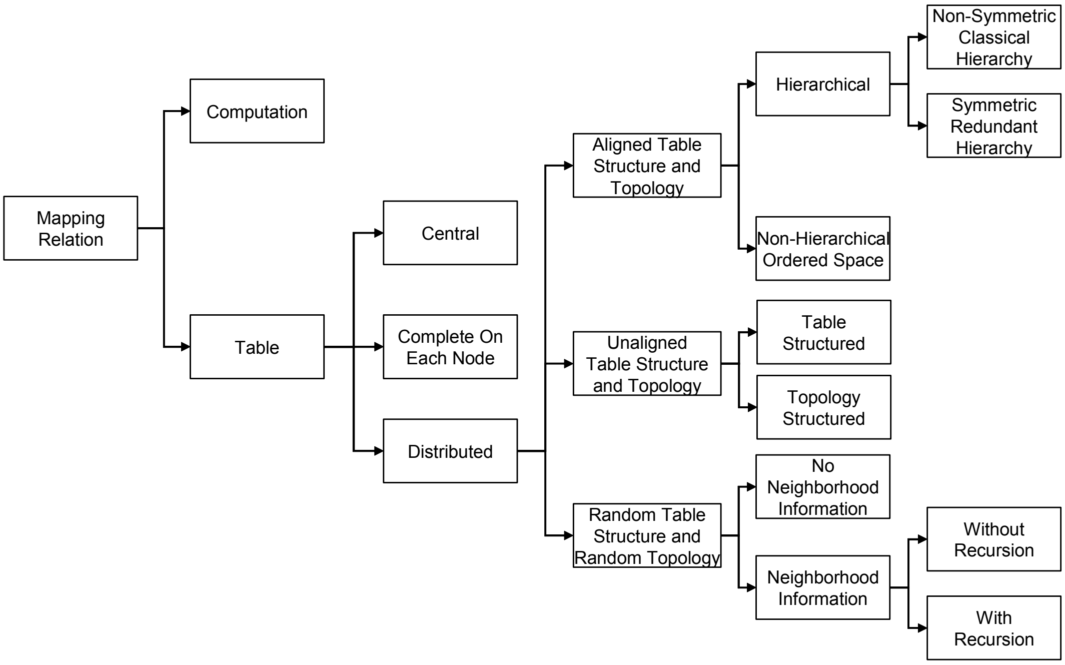
Mapping Relation
Computation
Table
Central
Complete On Each Node
Distributed
Aligned Table Structure and Topology
Unaligned Table Structure and Topology
Random Table Structure and Random Topology
Hierarchical
Non-Hierarchical Ordered Space
Non-Symmetric Classical Hierarchy
Symmetric Redundant Hierarchy
Table Structured
Topology Structured
No Neighborhood Information
Neighborhood Information
Without Recursion
With Recursion

Figure 3: Design Space for Mapping Relations in Distributed Systems

algorithms, "*symmetric hierarchies*" have been created by adding *redundancy.* In symmetric redundant hierarchies, every node can act as the root or be on any other level of the hierarchy. This can be achieved by replicating the root information on table areas on each node as well as the second-level information on sub-areas etc. Contemplate Figure 4 for more explanation. Let each position in the hierarchy be denoted by $P_{i_1, ..., i_{m<n}}$, where n is the number of levels and $i_1,...,i_m$ is the path of descendents below the root that leads to this position. The dark grey node of concern in the figure is then on position $P_{1,3}$. All nodes are associated with a unique position at the lowest level of the hierarchy and with all corresponding positions on the path up to the root. Hence, a node on position $P_{i_1, ..., i_n}$ maintains links to a complete set of nodes on positions $P_{i_1, ..., i_{n-1}, j}, P_{i_1, ..., i_{n-2}, j}, ..., P_{i_1, j}, P_j$, $j = 0, ..., b-1$, where b is the number of positions on each level (here assumed fix for simplification). In the figure, the dark grey node keeps replicas of the table information of the light grey positions and maintains links to a complete set of descendents. Note that nodes acting as a root will usually point to different neighbors for the second level table areas (and so on for all remaining levels) as there are multiple different options due to the replication.

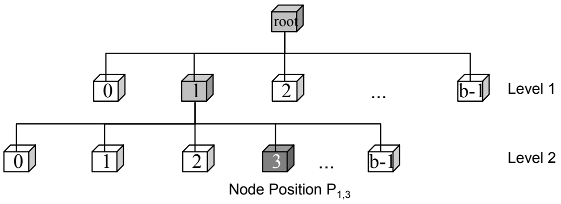

Figure 4: Symmetric Redundant Hierarchy

Non-hierarchical structures are also possible and available. In an *ordered space*, the table is split into consecutive areas. Each of the areas is represented on one node. The nodes, in turn, are ordered in the same way, *i.e.* neighboring table areas reside on neighboring nodes. Examples of such spaces are rings or Euclidean spaces, but other forms are possible.

Unaligned table structures and topologies occur when either the table is distributed according to a clear structure but the topology is random, or the topology is designed but the table structure random, or both table and topology are structured but in different ways. While the first case may be helpful to allow aggregation of table area information, the second case can be advantageous for performance improvements compared to a completely random approach. It appears difficult to gain from the third case.

Designs based on any kind of structured table regardless of the topology are sometimes referred to as *distributed hash tables*.

4 Two-Dimensional Design Space and Survey of P2P Search

This section presents a framework for design options in distributed or P2P search middleware and gives an overview of existing systems (cf. Table 1). By briefly discussing key requirements, advantages, and drawbacks, it explains the rationale for choosing a specific design. Furthermore, blind spots in the design space will be identified where further research may lead to entirely new systems with significantly improved performance. An extended discussion including non-P2P systems can be found in [8].

4.1 Computational Approaches

Computational mapping is very efficient in that it involves neither large tables to reside in memory nor bandwidth-consuming query messages to be sent. However, it is difficult to achieve as it requires that all possible outcomes of the computation be allowed in the target range, *i.e.* name space, address space, or space of routes. Constantly changing target

Table 1: Survey of Distributed and P2P Search in the Design Space

Design					Keyword Search	Lookup/Name Routing	Keyword Lookup/Routing
Computational					INS/Twine[a]	n/a	n/a
Table	*Central*				(Search engines, web directories)[b]	(Load balancing hub)[b]	Napster
	Complete on Each Node				Groove[a]	PGP key lookup, (NIC's Hosts.txt)[b]	n/a
	Distributed Table	*Aligned Table Structure and Topology*	*Hierarchical*	*Classical*	n/a	(DNS)[b]	TerraDir, Mutant Query Plans
				Symmetric Redundant	n/a	Pastry, Tapestry, AGILE, Kademlia/Overnet	SHARK
			Non-Hierarchical Ordered Space		n/a	CAN, Chord	Squid
		Unaligned Table Structure and Topology		*Table Structured*	n/a	(TRIAD/NBRP)[b]	n/a
				Table Unstruct., Topology Structured	n/a	HyperCuP[a]; Supernode networks like FastTrack (Morpheus, KaZaA, Grokster), Gnutella (Bear-Share Defender, Clip2 Reflector, LimeWire[c]), eDonkey	LimeWire[c], SIL
		Random Table Structure and Random Topology	*No Neighborhood Information*		n/a	Gnutella, Expanding Ring, Random Walk, Associative Overlays	Random Walk, Expanding Ring, LimeWire[c], Interest-based shortcuts
			Neighborhood Information	*Without Recursion*	(Manual http-Browsing)[b]	Freenet	Neurogrid
				With Recursion	n/a	Variants of Bloom filters	Bloom filters, e.g. LimeWire[c], PlanetP
Hybrid Systems					n/a	Yappers, Brocade	n/a

a. Only partial fit into category, see text for details

b. Not deployed for P2P

c. LimeWire proposes multiple add-ons to Gnutella and is subsequently listed multiple times in the table.

spaces or value ranges, due to the addition or removal of nodes and node addresses, or resources with their corresponding names, limit the applicability of computational approaches. Some have been made, however, usually involving hashing, and circumventing the problem by simply defining name or address spaces such as to cover all possible computation outcomes. This is impossible, however, for routing in dynamic environments, as the paths to nodes have to exist and cannot simply be defined.

INS/Twine [9] builds attribute-value trees from complex resource descriptions and disaggregates them into strands of variable length. Hashing is applied to map the strands onto 128-bit names. Chord's name routing algorithm completes the search.

4.2 Centralized Tables

Central tables are very bandwidth-efficient and incur little overhead. However, they require that a central entity have trust, reliability, and authoritative information access necessary to own the central table. Furthermore, a possible outage of a central server represents a considerable risk for the entire network.

Web search engines like Google apply inverse indices to provide URL names based on keywords. Many load balancing hubs route towards a specific server in a server farm based on the URL (name) of the request. Napster operates central servers to identify addresses of peers where content files with a file name containing given keywords are stored.

4.3 Completely Replicated Tables

Key advantages of *complete replication* of tables on each node are the increased autonomy and fault tolerance in the system when compared to central tables while keeping the simplicity and bandwidth-efficiency for requests. However, replication and synchronization issues as well as high memory needs restrict the approach to small tables and networks.

Collaboration tools like Groove synchronize keyword, name, and address information as well as actual objects or object updates on all nodes. Subsequently, all information necessary for, *e.g.*, keyword search, is available on all nodes, even though the system is based less on a reactive search but more on proactive synchronization. As a predecessor to the Domain Name System (DNS), the Network Information Center (NIC) distributed a file, hosts.txt, to all internet hosts for translation of domain names into IP addresses. For email encryption, PGP (pretty good privacy) connects a few thousand servers in a P2P fashion that store the public keys of PGP users. The servers synchronize via email so that every one of them has full knowledge of all keys.

4.4 Distributed Structured Tables with Aligned Topologies

Classical hierarchies are very efficient for searching and, in contrast to central tables, allow for delegation of responsibility. However, they require an equally hierarchical topology and source domain, *i.e.* keyword space, name space, or address space, in order to work efficiently.

DNS applies hierarchically organized domain names and an equivalent hierarchy of domain name servers to yield an IP address when asked about a domain name. TerraDir [10] organizes all content in a hierarchical keyword structure. For each content item or keyword, a virtual node is created, enabling keyword routing towards that node along the hierarchy. In [11], a multi-dimensional categorization hierarchy is managed by category servers and queries are processed by a hierarchy of meta index servers, index servers, and base servers. The multi-dimensionality enables searches along different criteria, like region, price, type of resource. Complex requests are parsed into 'mutant query plans' to allow for successive resolution of conjunct or adjunct queries or criteria within a request.

Symmetric redundant hierarchies combine the advantages of a classical hierarchy with the symmetry and fault tolerance requirements of a peer-to-peer system, at the cost of additional redundancy in the system and complex node and resource insertion and removal.

In Pastry [6] and Tapestry [5], content names and IP addresses of nodes are hashed onto the same numerical identifier (ID) space; this allows name routing when making that node responsible for holding a resource or a link to it that is closest to the resource in the ID space. The hierarchy is created through a digit representation of the ID to a base value and an association of each digit with one hierarchy level, starting from the last (Tapestry) or the first digit (Pastry), respectively. Kademlia [12], commercially deployed in Overnet, follows the same basic approach but uses a bit- (rather than digit-) representation of IDs to enable prefix matching via XORing bit strings. AGILE (Adaptive, Group-of-Interest-based Lookup Engine, [7]) follows a similar approach as Tapestry but introduces an additional three-level hierarchy for the resource description. Even though motivated through performance improvements (pruning), this is already a step towards symmetric redundant hierarchy-based keyword routing. This novel approach called SHARK has been found by identifying a blind spot in the design space and is currently being developed by the authors [13]. Here, the symmetric hierarchy concept is not applied to hash keys but to multidimensional keyword ontologies such as to accommodate scalable keyword routing along multiple adjunct or conjunct criteria, and it will be combined with a random structure further down the hierarchy to alleviate table and topology maintenance.

Also popular for name routing in peer-to-peer systems is the *non-hierarchical ordered space* approach. The prerequisite here is that source domain, *i.e.* keywords, names, or addresses, and nodes can be arranged in the same totally ordered, non-hierarchical space.

Chord [3] hashes resource names and node IP addresses to a 128-bit ID. The IDs are arranged in a circle with the predecessor node of a resource ID being responsible for providing the resource or a link to it. Fingers are used as short-cuts to prevent the name routing mechanism from moving around the circle in unit steps. In CAN (Content Addressable Network, [4]), hashing is similarly applied to map resource names onto an ID in a d-dimensional torus. Nodes distribute responsibility for the ID space among themselves and maintain virtual links to all direct neighbors in the torus. Queries for a name, *i.e.* ID, can then at each node easily be routed into the best direction. Squid [14] assumes a d-dimensional space of all allowed keywords, lexicographically ordered in each dimension. A Hilbert space-filling-curve is used to reduce the dimensionality to one while preserving locality of ranges. A variant of Chord is constructed for keyword routing in the remaining dimension.

4.5 Unaligned Distributed Table Structures and Topologies

Search based on *unaligned table structures and topologies* is most common where a structuring of the table appears prohibitive yet a structured topology improves system performance. This particularly applies to widely-spread peer-to-peer systems, where free-riding and non-trustworthiness of some peers inhibit distributing responsibility for some table entries beyond the corresponding resources' owner.

HyperCuP builds a hypercube topology of nodes to support efficient flooding [15]. An extension arranging resources in the same hypercube space as the nodes which is also proposed by the authors would move it to ordered-space keyword routing. Hierarchies with "supernodes" or "ultranodes" [16] are introduced into many peer-to-peer name routing systems to improve scalability, like in FastTrack (Morpheus, Kazaa, Grokster) or Gnutella applications (LimeWire, BearShare Defender, Clip2 Reflector) [17]. In these systems, the supernodes replicate address and name information for all subordinate nodes and act as gateways or proxies for name requests between the subordinate nodes and the remaining network. However, as only a hierarchical overlay topology is applied, but not an equivalently hierarchical address or name space, supernodes still have to flood queries among them. eDonkey also applies supernode-like servers which are, however, not connected to one another. It is the clients' task to successively post requests to all servers they know. Limewire also proposes rich XML-based keyword routing on a supernode organization. A general framework for designing topologies for keyword routing without structuring the distributed table, SIL (search/index links) is presented in [18] and leads to the proposition of parallel clusters rather than supernodes.

TRIAD/NBRP (Translating Relaying Internet Architecture integrating Active Directories/Name-Based Routing Protocol, [19]), in contrast, builds on a *hierarchical name space* like DNS in order to aggregate content or name information, and a natural hierarchy of servers responsible for certain names. However, the *topology* between routers can be *random*. Routers simply advertise reachability of certain name suffixes to allow for efficient name routing.

4.6 Random Distributed Tables and Random Topologies

Even though *random table structures* for mapping relationships in search appear less sophisticated than aligned structures and do not allow to exploit implicit structural knowledge, there are a couple of advantages to this approach. The maintenance burden for creating and keeping an explicit table structure can be too high, particularly in fast-changing environments like some P2P or, even more so, mobile ad-hoc networks. This also leads to issues regarding fault-tolerance: if the structure is not correct in algorithms that rely on it, queries may not be successful. Finally, structured approaches require a high degree of collaboration and trust. Unless ownership for certain resources referred to in table entries coincides with the assigned responsibility for these table entries, resource owners and search requestors have to rely on third parties to provide correct information. Even though this coincidence applies for DNS, in many cases, like Pastry or Tapestry, this is not the case.

The most simple form of a randomly distributed table lets each node only maintain a table of keywords, names, and addresses (if several on a node) of its *own resources without neighborhood information*. This approach is extremely simple and helpful in environments changing so fast that knowledge about neighbors becomes stale before it is used. However, for all mappings, it requires to either more or less arbitrarily choose neighbors to send requests to, or, more commonly, flood the entire network.

The approach is used in Gnutella name routing and its extension to keyword routing proposed within LimeWire. In expanding ring searches, the requestor is contacted before each additional request forwarding to check whether the desired object has already been found such as to allow early termination of the query flooding. Multiple random walks with termination checking can drastically reduce the number of messages due to the finer granularity of node visits and reduced duplication of messages, and, hence, improve bandwidth scalability [20]. The improvements, however, require sufficient replication of objects and come at the cost of increased latency. Interest-based shortcuts [21] can be created in arbitrary topologies based on past successful responses to exploit interest locality and support semantic clustering. Similarly, guide rules are proposed in [22] to create associative overlays and limit flooding to peers who have at least one item in common with the requestor.

Direct neighborhood information on each node can improve query forwarding decisions within a distributed table. However, unless flooding is used, the approach remains indeterministic as to whether a result can be found in the direction of a neighbor.

Hyperlinks in http provide users with names, *i.e.* URLs, of resources on neighboring nodes; they can be used for manual keyword search or browsing. In Freenet [23], name routing is based on hashes of file names or content information. Each node forwards a query to the neighbor storing content with a hash ID numerically closest to the request. The approach converges due to Freenet's aggressive caching strategy. Similarly, in Neurogrid [24], the same approach is used for keyword meta-data rather than hash IDs.

Recursive neighborhood information, usually in a very aggregated form, makes a random table structure search more deterministic while avoiding flooding. However, the synchronization overhead incurred can be substantial. Specific attention is due in networks containing loops in order to avoid a count-to-infinity problem.

For name and keyword routing in peer-to-peer networks, various variants of Bloom filters have been proposed to aggregate and compress information on resources in the direction of each neighbor. Put simple, one bit is set in a word for each name occurring in a certain direction. Rhea and Kubiatowicz suggest attenuated Bloom filters storing name information up to d-levels of depth with weights decreasing with distance [25]. For keyword routing, Prinkey proposes standard Bloom filters in tree topologies with aggregated signatures of a branch, *i.e.* the Bloom filter bits represent the hashed keywords present in a tree branch. LimeWire modifies the proposal to cope with arbitrary topologies by adding the number of hops to a resource when propagating the keyword routing information. Crespo and Garcia-Molina [26] suggest to store and propagate the number of matching documents for each keyword, either together with the number of hops to a document, or weighting the number of documents with a cost function depending on the distance. In PlanetP [27], nodes build inverted indices of keywords for the objects they hold locally and summarize them through Bloom filters. Gossiping is used to propagate this information and develop knowledge of the surrounding network. PlanetP resolves queries by ranking the relevance of neighbors according to the angle between a weighted vector space representation of the query and the Bloom filter profiles. The system can scale to a vast number of documents but only a few thousand participating nodes.

4.7 Hybrid Approaches

Apart from the approaches presented above, it is, of course, possible to combine different approaches within one *hybrid* system to try and reap the benefits from multiple systems.

In Brocade [28], all (potentially unstable or high-latency) peers within a neighborhood connect to one central stable high-bandwidth landmark, to which they transfer their indices and post requests. The landmarks in turn are then well suited to form a Tapestry-based distributed hash table for requests beyond a neighborhood.

Yappers [29] forms a rough symmetric redundant hierarchy with only one level of b buckets, while it applies a random topology and table structure beyond the first level, *i.e.*, within a bucket. The symmetric redundant hierarchy itself is not explicitly managed as usual but obtained by choosing a large enough immediate random neighborhood for publishing and query initiation so that nodes within all first-level buckets are present with high probability, and by assigning secondary buckets to nodes if necessary. An extended neighborhood is maintained so that flooding within a bucket can extend beyond the immediate random neighborhood.

5 Evaluation of the Design Options

The best design alternative to choose for specific middleware depends on the target application and its requirements. This section gives a generic evaluation of the design options along a set of requirements.

5.1 Functional Design Space

The functional design space can be condensed into two major choices: (a) build a disaggregated search involving the separate steps keyword search and lookup/name routing or (b) take an integrated keyword routing approach. This is slightly more complex in the non-P2P world, please refer to [8] for a discussion.

Integrated Approaches

Integrated approaches avoid a duplication of highly similar mapping functionalities and are thus more *efficient*. Particularly in widely distributed tables (rather than computational, centralized, or strictly hierarchical approaches), each mapping requires the collaboration of many nodes and incurs high *bandwidth demands* and *latency*.

Integrated keyword routing also allows *keyword-based rerouting* and, hence, makes the system transparent to name changes, *e.g.*, due to addition or removal of content for a given keyword. This can also alleviate *real-time search*.

Disaggregated Approaches

A decoupling of keyword search and lookup/name routing shows the set of following advantages:

- *Reusability*: Each mapping can be used separately to support a wider range of services; *e.g.*, name routing can not only be used for keyword search but also for data retrieval in a distributed backup service.
- *Innovation*: Innovation in one area does not affect another area; *e.g.,* improvements to search or string matching algorithms are independent of any changes to the name routing software.
- *Horizontal and Vertical Variety*: Different choices for keyword search and lookup/name routing smoothly interoperate, both horizontally at each step, *e.g.*, centralized Google-type keyword search in parallel to decentralized P2P search like Infrasearch, as well as vertically across the two steps, *e.g.*, central search and hierarchical lookup/name routing.

Overall, integrated keyword routing should usually be the method of choice for P2P systems, particularly as the following advantages of disaggregated approaches in non-P2P scenarios do not apply:

- *Ownership Separation*: Each mapping can be offered by a separate entity, allowing more competition leading to higher efficiency and innovation, potentially also hampering censorship. In true P2P, however, every peer participates in all mappings anyway.
- *Delegation*: Almost a consequence of ownership separation and vertical variety, but extremely important, is the possibility to logically separate keyword space, name space, and address space, *e.g.*, independent allocation of and even delegation of responsibility for IP address space and domain names. P2P systems, however, do not (or not yet) apply any (hierarchical) structures like domains.
- *Simplicity*: Devices for each step can be simpler and more specialized than for integrated systems, *e.g.*, ultra-fast hardware-based IP routers. Similarly, software engineering is alleviated, also increasing the *maintainability* of the system. The devices in P2P systems are, however, usually general purpose computers and not specialized.

5.2 Structural Design Space

Table 2 presents an overview of system design requirements and the degree to which they are met by various structural middleware design options. Note that many system developers have added more details to their design-specific features that address shortcomings or fortify strengths of the basic approach; furthermore, unaligned structures and topologies and hybrid approaches are not considered as they can be too diverse. Table 2 can thus only be regarded as a rough guideline. Major advantages and prerequisites of each system have already been highlighted in Section 4. An explanation and definition of these requirements is given below.

- *Manageability and Control*: How hard is it to control and manage the system, *i.e.* how complex is it, how much maintenance does it require, and what level of control can be exercised?
- *Coherence*: Does the system deterministically find authoritative information, does it behave in an indeterministic way, or is it prone to retrieve stale replicas?
- *Extensibility*: How easy or difficult is it to add resources or nodes to the system?
- *Fault Tolerance and Adaptability*: How severely is the system affected by a fault and how easily and quickly can it cope with system changes like node joins or leaves?
- *Scalability*: To what limits can the system grow at reasonable performance, particularly with regards to bandwidth and latency, but also memory and processing load on average nodes as well as hot spots?
- *Publish Autonomy and Security*: Is information mostly kept at the resource owner, can even responsibility for the corresponding name or address space be delegated?
- *Search Autonomy and Security*: Does searching require the collaboration of many, particularly untrusted parties?
- *Infrastructure Independence*: How independent is the system from shared infrastructure like central servers? P2P networks usually try to avoid central infrastructure.
- *Special Prerequisites*: As already discussed in Section 3, some designs have special requirements. A computational approach requires that all computation results lead to valid values in the target range of the mapping, *i.e.* name space, address space (or space of paths in non-P2P). Hierarchical approaches require the source domains of the mapping, *i.e.* keyword space, name space (or address space in non-P2P), to be hierarchical. Finally, a non-hierarchical ordered-space approach requires an arbitrary order other than a hierarchy to be imposed on the source domain, usually a linear order.

Table 2: Evaluation of Structural Design Options

Design					Manageability and Control	Coherence	Extensibility	Fault Tolerance & Adaptability	Scalability	Publish Autonomy and Security	Search Autonomy and Security	Infrastructure needs	Special Prerequisites[a]
Computational					high	easy	none	none	high	full	full	none	yes
Table	*Central*				high	easy	low	low	high[b]	low/high[c]	low/high[c]	server	no
	Complete on Each Node				low	diff.	mod.	high	v. low	full	full	none	no
	Distributed Table	*Aligned T.S & Topology.*	*Hierar-chical*	*Classical*	mod./ high[d]	mod./ easy[d]	mod./ high[d]	low	high[b]	low/high[cd]	low/high[c]	server hierarch.	yes
				Symmetric	low/ mod.[d]	diff./ mod.[d]	mod./ high[d]	mode-rate	high	low/high[d]	low	none	yes
			Non-Hierarchical Ordered Space		low/ mod.[d]	diff./ mod.[d]	mod./ high[d]	mode-rate	high	low/high[d]	low	none	yes
		Random T.S. & Random Topology	*No Neighborhood Information*		low	easy	very high	very high	very low	full	low	none	no
			Neighbor. Info.	*Without Recursion*	low	mode-rate	high	high	low	high	low	none	no
				With Recursion	low	mode-rate	high	mode-rate	mode-rate[e]	moderate	moderate	none	no

a. See text for explanation
b. If additional servers added
c. If central entities regarded as trusted and collaborative
d. Usually table structure and topology are different from ownership structure. Second (better) value applies if they are aligned (like in DNS)
e. Depending on level of information aggregation and rate of information updates

6 Summary, Conclusions, and Future Work

A design space that is believed to be exhaustive has been developed to classify current and facilitate future research in P2P search middleware. It outlines the application-relevant differences between the possible approaches based on a break-down of options along two basic dimensions of distributed search design: the functional and the structural dimension. A classification and description of example systems validated the framework and serves as a survey of P2P search systems.

The trade-offs made and effects of choosing one design over another should be evaluated with respect to the most relevant requirements and criteria of an application in both dimensions, functional and structural. A set of requirements and a high-level assessment of design options along these requirements has been presented and forms the basis for this evaluation.

Based on the design space, a novel search design for peer-to-peer systems, keyword routing based on symmetric hierarchies, has been identified. The corresponding system, SHARK, has been outlined in this document and may lead to significant performance improvements for some applications.

Going forward, further discussions will clarify additional implications of the design space and enable an assessment of general trends in P2P search middleware research from a broad perspective.

Acknowledgements

This work has been performed partially in the framework of the EU IST project MMAPPS "Market Management of Peer-to-Peer Services" (IST-2001-34201), where the ETH Zürich has been funded by the Swiss Bundesministerium für Bildung und Wissenschaft BBW, Bern under Grant No. 00.0275.

References

[1] S. Saroiu, P. Gummadi, S. Gribble, "Exploring the Design Space of Distributed and Peer-to-Peer Systems: Comparing the Web, TRIAD, and Chord/CFS," *1st Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, U.S.A., Mar. 2002.

[2] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica, "Looking up data in P2P systems," *Comm. of the ACM*, vol. 46, no. 2, Feb. 2003.

[3] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, I. Stoica, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," *ACM SIGCOMM*, San Diego, CA, U.S.A., Aug. 2001.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content-Addressable Network," *ACM SIGCOMM*, San Diego, CA, U.S.A., Aug. 2001.

[5] B. Zhao, J. Kubiatowicz, A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Technical Report UCB/CSB-01-1141*, Computer Science Division, U.C. Berkeley, CA, U.S.A., Apr. 2001.

[6] Druschel, Rowstron, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," *IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov. 2001.

[7] J. Mischke, B. Stiller, "Peer-to-Peer Overlay Network Management Through AGILE," *IFIP/IEEE Int'l Symposium on*

Integrated Network Management (IM), Colorado Springs, CO, U.S.A., Mar. 2003.

[8] J. Mischke, B. Stiller, "Design Space for Distributed Search $(DS)^2$ - A System Designers' Guide," *ETH-Zurich*, Switzerland, TIK Report Nr. 151, Sep. 2002.

[9] M. Balazinska, H. Balakrishnan, D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," *Pervasive 2002 - Int'l Conf. on Pervasive Computing*, Zurich, Switzerland, Aug. 2002.

[10] B. Silaghi, S. Bhattacharjee, P. Keleher, "Routing in the Terra-Dir Directory Service," *SPIE ITCOM'02*, Boston, MA, U.S.A., Jul. 2002.

[11] V. Papadimos, D. Maier , K. Tufte, "Distributed Query Processing and Catalogs for Peer-to-Peer Systems," *Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, U.S.A., Jan. 2003.

[12] P. Maymounkov, D. Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," *1st Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, U.S.A., Mar. 2002.

[13] J. Mischke, B. Stiller, "An Efficient Protocol Specification, Implementation, and Evaluation for a Highly Scalable Peer-to-Peer Search Infrastructure," *To appear, 9th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Seoul, Korea, Apr. 2004.

[14] C. Schmidt, M. Parashar, "Flexible Information Discovery in Decentralized Distributed Systems," *12th Int'l Symposium on High-Performance Distributed Computing (HPDC)*, Seattle, WA, U.S.A., Jun. 2003.

[15] M. Schlosser, M. Sintek, S. Decker, W. Nejdl, "HyperCuP - Hypercubes, Ontologies and Efficient Search on P2P Networks*," Int'l Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, Bologna, Italy, Jun. 2002.

[16] B. Yang, H. Garcia-Molina, "Designing a Super-peer Network,"*19th Int'l Conf. on Data Engineering (ICDE)*, Bangalore, India, Mar. 2003.

[17] K. Truelove, A. Chasin, "Morpheus Out of the Underworld," Jul. 2002, http://www.openp2p.com/pub/a/p2p/2001/07/02/morpheus.html (Aug. 29, 2002).

[18] B. Cooper, H. Garcia-Molina, "Studying search networks with SIL," *2nd Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, U.S.A., Feb. 2003.

[19] M. Gritter, D. Cheriton, "An Architecture for Content Routing Support in the Internet," *3rd Usenix Symposium on Internet Technologies and Systems (USITS)*, San Francisco, CA, U.S.A., Mar. 2001.

[20] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, "Search and replication in Unstructured Peer-to-Peer Networks," *16th ACM Int'l Conf. on Supercomputing (ICS'02)*, New York, U.S.A., Jun. 2002.

[21] K. Sripanidkulchai, B. Maggs, H. Zhang: "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *INFOCOM'03*, San Francisco, U.S.A., Apr. 2003.

[22] E. Cohen, A. Fiat, H. Kaplan, "A case for associative Peer to Peer Overlays," *HotNets-I*, Princeton University, Princeton, NJ, U.S.A., Oct. 2002.

[23] I. Clarke, S. Miller, T. Hong, O. Sandberg, B. Wiley, "Protecting Free Expression Online with Freenet," *IEEE Internet Computing*, vol. 6, no. 1, Jan./Feb. 2002.

[24] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," *Int'l Workshop on Peer-to-Peer Computing*, Pisa, Italy, May 2002.

[25] S. Rhea, J. Kubiatowicz, "Probabilistic Location and Routing," *INFOCOM'02*, New York, U.S.A., Jun. 2002.

[26] A. Crespo, H. Garcia-Molina, "Routing Indices For Peer-to-Peer Systems," *Int'l Conf. on Distributed Computing Systems (ICDCS)*, Vienna, Austria, Jul. 2002.

[27] F. Cuenca-Acuna, C. Peery, R. Martin, T. Nguyen, "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities," *12th Int'l Symposium on High-Performance Distributed Computing (HPDC)*, Seattle, WA, U.S.A., Jun. 2003.

[28] B. Zhao, Y. Duan, L. Huang, A. Joseph, J. Kubiatowicz, "Brocade: Landmark Routing on Overlay Networks," *1st Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, U.S.A., Mar. 2002.

[29] P. Ganesan, Q. Sun, H. Garcia-Molina, "YAPPERS: A Peer-to-Peer Lookup Service Over Arbitrary Topology," *INFOCOM'03*, San Francisco, U.S.A., Apr. 2003.

Jan Mischke [mischke@tik.ee.ethz.ch] is a PhD student at ETH Zurich and a strategic consultant with McKinsey&Company. He studied Electrical and Electronic Engineering in Aachen, RWTH, and London, Imperial College, and received the "Diplom Ingenieur" (M.Sc.) from RWTH in 2000. He received numerous awards, including the Friedrich-Wilhelm prize, the Philips prize, and the Springorum medal. His main research interest are P2P and overlay networks.

Prof. Dr. Burkhard Stiller [stiller@tik.ee.ethz.ch, stiller@informatik.unibw-muenchen.de] received his German diploma degree in computer science and his doctoral degree from the University of Karlsruhe, Germany in 1990 and 1994, respectively, where he has been a Research Assistant at the Institute of Telematics, University of Karlsruhe. He was on leave at the University of California, Irvine, and the University of Cambridge, Computer Laboratory, England. Currently, he is an Assistant Professor for Communication Systems at the ETH Zurich, Switzerland and a Full Professor for computer science at the University of Federal Armed Forces Munich, Germany in the Department of Computer Science. His areas of interest include Internet communications, charging and accounting for packet-based services, pricing schemes, QoS, mobility and AAA architectures, and peer-to-peer systems. Burkhard Stiller is member of the editorial board of the Kluwer's Netnomics Journal on economic research and electronic networking, the ACM, and the German Society for Computer Science GI. Besides being the PC Co-chair of the IEEE/IFIP DSOM Workshop 1999, the ICQT'01 Workshop, he was General Chair of the QofIS/ICQT'02 and NGC/ICQT'03 Workshops. He will be the General Chair for the LCN'04 conference in Tampa, Florida, and PC Co-chair of ICQT'04.

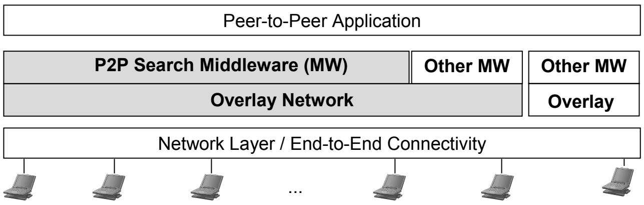
Peer-to-Peer Application
P2P Search Middleware (MW)
Other MW
Other MW
Overlay Network
Overlay
Network Layer / End-to-End Connectivity
...

Figure 1: P2P Networks: Systems View and Layering

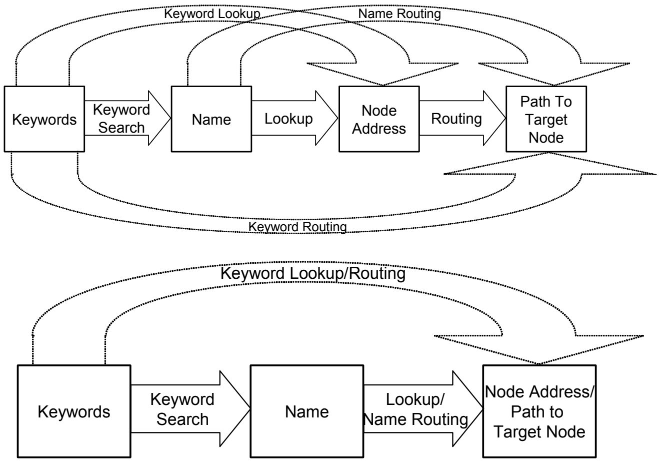

Figure 2: Search in Distributed (top) and P2P (bottom) Systems

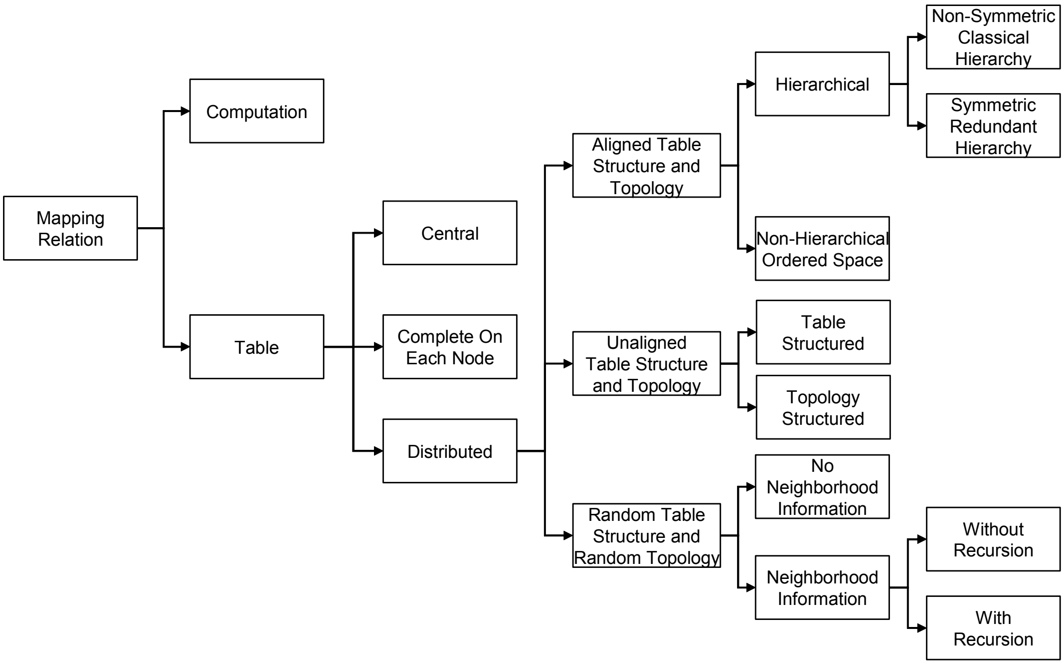
Mapping Relation
Computation
Table
Central
Complete On Each Node
Distributed
Aligned Table Structure and Topology
Unaligned Table Structure and Topology
Random Table Structure and Random Topology
Hierarchical
Non-Hierarchical Ordered Space
Non-Symmetric Classical Hierarchy
Symmetric Redundant Hierarchy
Table Structured
Topology Structured
No Neighborhood Information
Neighborhood Information
Without Recursion
With Recursion

Figure 3: Design Space for Mapping Relations in Distributed Systems

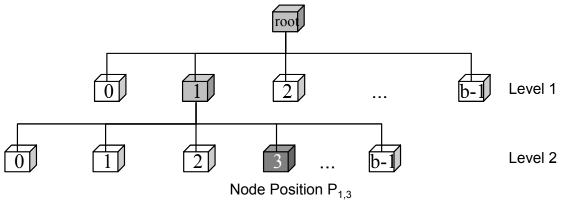

Figure 4: Symmetric Redundant Hierarchy

Table 3: Survey of Distributed and P2P Search in the Design Space

Design					Keyword Search	Lookup/Name Routing	Keyword Lookup/Routing
Computational					INS/Twine[a]	n/a	n/a
Table	*Central*				(Search engines, web directories)[b]	(Load balancing hub)[b]	Napster
	Complete on Each Node				Groove[a]	PGP key lookup, (NIC's Hosts.txt)[b]	n/a
	Distributed Table	*Aligned Table Structure and Topology*	*Hierarchical*	*Classical*	n/a	(DNS)[b]	TerraDir, Mutant Query Plans
				Symmetric Redundant	n/a	Pastry, Tapestry, AGILE, Kademlia/Overnet	SHARK
			Non-Hierarchical Ordered Space		n/a	CAN, Chord	Squid
		Unaligned Table Structure and Topology		*Table Structured*	n/a	(TRIAD/NBRP)[b]	n/a
				Table Unstruct., Topology Structured	n/a	HyperCuP[a]; Supernode networks like FastTrack (Morpheus, KaZaA, Grokster), Gnutella (Bear-Share Defender, Clip2 Reflector, LimeWire[c]), eDonkey	LimeWire[c], SIL
		Random Table Structure and Random Topology	*No Neighborhood Information*		n/a	Gnutella, Expanding Ring, Random Walk, Associative Overlays	Random Walk, Expanding Ring, LimeWire[c], Interest-based shortcuts
			Neighborhood Information	*Without Recursion*	(Manual http-Browsing)[b]	Freenet	Neurogrid
				With Recursion	n/a	Variants of Bloom filters	Bloom filters, e.g. LimeWire[c], PlanetP
Hybrid Systems					n/a	Yappers, Brocade	n/a

a. Only partial fit into category, see text for details

b. Not deployed for P2P

c. LimeWire proposes multiple add-ons to Gnutella and is subsequently listed multiple times in the table.

Table 4: Evaluation of Structural Design Options

Design					Manageability and Control	Coherence	Extensibility	Fault Tolerance & Adaptability	Scalability	Publish Autonomy and Security	Search Autonomy and Security	Infrastructure needs	Special Prerequisites[a]
Computational					high	easy	none	none	high	full	full	none	yes
Table	*Central*				high	easy	low	low	high[b]	low/high[c]	low/high[c]	server	no
	Complete on Each Node				low	diff.	mod.	high	v. low	full	full	none	no
	Distributed Table	*Aligned T.S & Topology.*	*Hierar-chical*	*Classical*	mod./ high[d]	mod./ easy[d]	mod./ high[d]	low	high[b]	low/high[cd]	low/high[c]	server hierarch.	yes
				Symmetric	low/ mod.[d]	diff./ mod.[d]	mod./ high[d]	mode-rate	high	low/high[d]	low	none	yes
			Non-Hierarchical Ordered Space		low/ mod.[d]	diff./ mod.[d]	mod./ high[d]	mode-rate	high	low/high[d]	low	none	yes
		Random T.S. & Random Topology	*No Neighborhood Information*		low	easy	very high	very high	very low	full	low	none	no
			Neighbor. Info.	*Without Recursion*	low	mode-rate	high	high	low	high	low	none	no
				With Recursion	low	mode-rate	high	mode-rate	mode-rate[e]	moderate	moderate	none	no

a. See text for explanation

b. If additional servers added

c. If central entities regarded as trusted and collaborative

d. Usually table structure and topology are different from ownership structure. Second (better) value applies if they are aligned (like in DNS)

e. Depending on level of information aggregation and rate of information updates