A Survey on Resource Discovery Mechanisms, Peer-to-Peer and Service Discovery Frameworks.

Elena Meshkova, Janne Riihijärvi, Marina Petrova and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {eme, jar, mpe, pma}@mobnets.rwth-aachen.de

***Abstract*—Service and resource discovery has become an integral part of modern networked systems. In this survey we give an overview of the existing solutions for service and resource discovery for a wide variety of network types. We cover techniques used in existing systems, as well as recent developments from the research front. We also provide taxonomy for discovery systems and architectures, and review the various algorithms and search methods applicable for such systems. Peer-to-peer overlays are discussed in detail and solutions for non-IP-based networks are also included in the review. We also specifically comment on issues related to wireless networks, and give an overview of the various issues and complications that should be considered in future work in this domain.**

***Index Terms*—service discovery, resource discovery, overview, survey, search, overlay, types of clustering, peer-to-peer, DHT, Kademlia, Pastry, Napster, Chord, Gnutella, Viceroy, CAN, KaZaA, FastTrack, eDonkey, Freenet, BitTorrent, bootstrapping, resource constrained, wireless networks, ad hoc, heterogeneous networks, ubiquitous computing, sensor networks, UPnP, SLP, Salutation, Bonjour, Jini.**

I. Introduction

Service and resource discovery is becoming more and more important with the growth in the size and the diversity of computer networks. Furthermore, the ubiquity of the mobile and wireless networks is making different discovery services critically important in the future. Considerable amount of work has been done in this field, but mostly the resource and service discovery solutions have been approached as an implementation task to develop new protocols or frameworks, not trying to classify, categorize and to seek out generalities.

This survey paper covers resource and service discovery (SD) mechanisms in general, provides a taxonomy to differentiate between various systems and describes several popular service discovery and peer-to-peer frameworks. This survey paper covers resource and service discovery mechanisms in general, and we also provide a taxonomy to differentiate between considered systems. There have been several recent review type of articles, among them the ones by Edwards [1], Zhu *et al.* [2], Vanthournout *et al.* [3] and K. Lua *et al.* [4]. The scope of our review has a wider scope than the ones in [1] or [2]. The work of E. Lua *et al.* [4] covers P2P networks, but the traditional service discovery systems are out of scope of the paper. The article [3] is, in part, closer to our approach, but is limited to IP-based resource discovery mechanisms, and focuses mostly on taxonomy development. We will consider also non-IP systems, and we are technology agnostic between P2P, wireless and large-scale Internet based frameworks. We will also present some popular search algorithms and shortly comment on issues related to wireless and embedded networks in the resource discovery context.

The paper is organized as follows. In the rest of the introduction we provide the basic terminology and main characterization of the service discovery frameworks. In the Section II the possible system architectures are described and classified with more details. We discuss different network related issues like packet propagation and dynamic query termination techniques in Section III. In Section IV the distributed search algorithms and protocols that utilize these methods are introduced. The discussed search protocols are mostly applicable to unstructured decentralized system. Different clustering approaches, that are used in hybrid frameworks, are addressed in Section V. In Section VI we discuss decentralized structured architectures are analyzed on the example of DHT-based system. The most famous and well-spread peer-to-peer systems are discussed in Section VII. In Section VIII we describe classical service discovery frameworks. The issues specific to service discovery in non-fixed and heterogeneous networks are discussed in Section IX. Additionally, in Section X we describe some complications to be considered when designing a service discovery system. Finally, in Section XI we provide some general discussions and conclusions.

A. Terminology and Main Characteristics

The terms *service discovery* and *resource discovery* are often used interchangeably. Although, some subtle differences could be defined on this, we do not to take a firm stand on the issue. Oxford English Dictionary [5] gives the following definition:

> *Resource (n.):* A means of supplying some want or deficiency; a stock or reserve upon which one can draw when necessary; an action or procedure to which one may have recourse in a difficulty or emergency; an expedient device, shift.

In general, we follow Vanthournout *et al.* [3] by defining that a resource is any source of supply, and specifically can consist of files, file-system, memory, CPU-capability, communication capability (e.g., radio modem), etc. *Resource discovery* is any mechanism that is providing capability to locate resources. We, furthermore, loosely define *service discovery* as a subset of resource discovery, in such a way that service discovery should be mostly seen as a capability to find specific services such as applications or well-defined networked services that are not pure abstractions. However, the difference between service and resource discovery is not particularly important, or interesting, for this survey.

The specific terminologies used to describe resource and service discovery frameworks[1] are often highly different, in part because this gives the possibility to emphasize different and unique aspects in their designs. A taxonomy of service discovery systems featuring the commonly used terminology is given in Figure 1.

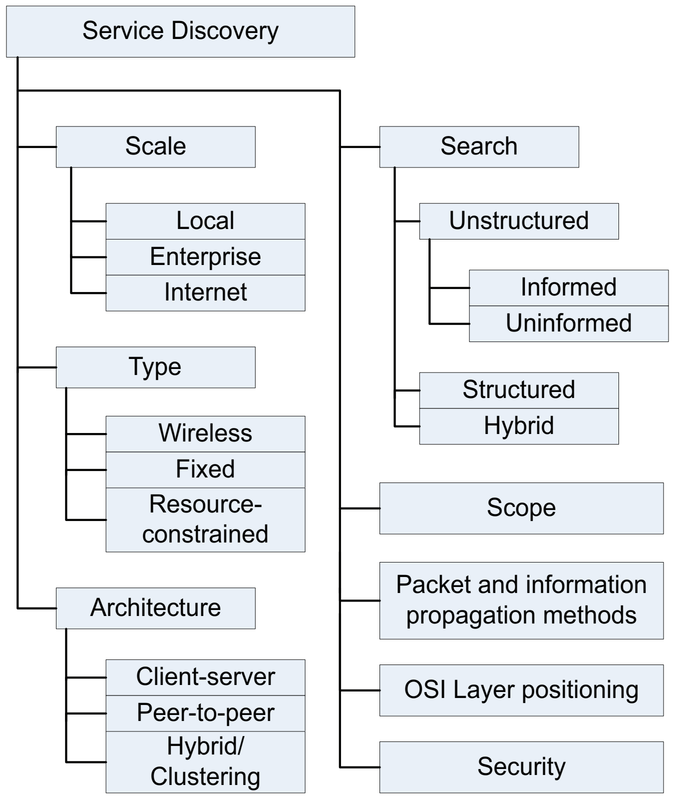


Fig. 1. The overview of the service discovery taxonomy.

An important issue is the *naming* of resources. With naming we denote any mechanism that supports a logical way to give and maintain resource names in service discovery systems. General naming systems or schemes are out of the scope of this article. An integral part of the resource discovery process is to search matching resources for client requests. Both naming and *search* are, of course, generic problems that are encountered in many computer science and networking applications. A number of different resource naming approaches have been proposed. The selection can strongly affect and limit the discovery mechanisms. The basic approaches can be roughly divided to *hash table*, *attribute/string*, and *directory* based naming. The alternative taxonomy is based on *template-based*, *predefined template* and *free-form* approaches.

[1]We use here specifically term *framework* to emphasize that we do not limit us to resource discovery *protocols*, but consider also system level proposals.

B. Main characteristics of the system

Each service discovery (SD) system has either structured or unstructured *architecture*. Structured architectures are further subdivided into centralized (client-server) or decentralized ones. Hybrid architectures try to combine the advantages offered by different architecture types to boost the overall system performance. The original Gnutella [6] follows the unstructured approach while SLP [7] and Napster [8] are examples of structured centralized systems. DHT-based systems, for example Kadmelia [9] based eDonkey networks called Overnet and Kad, are the examples of decentralized structured systems. Gnutella2 [10] or KaZaa/FastTrack [11] are both based on hybrid architectures.

SD systems use a variety of *search* techniques to discover a service. These range from simple flooding or unicasting to the use of such sophisticated methods as simulated annealing or genetic algorithms. We emphasize difference between *search techniques* and *queries and query matching*. The search mechanism is clearly focused on how to distribute the queries into the network and how the services are found. Although the advanced methods attempt to intelligently decide how the queries are distributed in the search phase, the fundamental problem is the optimization and coverage of the communications. Queries and query matching are on other hand all about how to make proper queries and how the user (client) queries are matched against the existing resources. This problem is more related to issues such as parsing, language definitions, directory structures, hash-table management, etc. The capability to *resolve (complex) queries* is, typically, a strongly differentiating factor for SD systems. Simple attribute or hash based systems are relatively limited in their expression power. The unstructured or centralized systems allow much higher query expressiveness. However, *fuzzy matching*, i.e. the ability of SD systems to provide answers to partial or incomplete queries, as well as the resolution of *free form queries*, that allow much richer query formats and extendibility than the current simple parsing and keyword matching algorithms, is beyond the capabilities of most discovery frameworks. Query matching, overall, is out of scope of our review and we provide just general information on this topic. More detail information can be found for example in [12]. We concentrate on search mechanisms and make exception to this in the case of systems in which the query structure is closely related to the search mechanisms used, as is the case of DHT-based systems (Section VI).

The studied systems are also designed to be used with different network *topologies*. Most of the existing SD-systems are designed for fixed networks. However, there are an increasing number of proposals [13], [14], [15] that are specifically designed for ad-hoc networks, sensor or hybrid networks. Topology and mobility are important issues as they can strongly influence to the performance of SD.

The design of service discovery systems depends on the *scale* of their deployment. The frameworks created to work on *internet-scale* (over a million users) require the highest scalability. Most of such systems are hybrid P2P overlays, such as eMule [16], Freenet [17] or FastTrack [11]. The DNS relies on a tree-structured organization of servers. Web-search systems, like Google, rely on "nests" of servers situated all round the world that resolve user queries. The Internet-scale systems typically allow to search for different files and other web-resources.

Enterprise-scale systems such as SLP [7] or Jini [18] have less servers (directory agents) and use simpler protocols to operate. These systems typically have a client-server infrastructure and impose high demands for security. The provided services range from a search for particular documents to printing and obtaining a particular sensor reading.

Local area networks and personal area networks due to their size utilize the simplest architectures. Typically they impose high performance demands on a discovery system. A well-known UPnP [19] is an example of such a framework. The small-scale systems aim to provide the whole range of services that a user can possibly have in his home or office. Overall we can conclude that *the complexity of the SD systems grows either with its size or with the increase in performance demands or the environment constraints*.

Service discovery systems also differ in *scope*. Some, like Google [20] or KaZaa [21], allow to search mostly for data and other web resources. Others, like UPnP or Bonjour [22], discover various services and devices in the neighborhood, and could be easily extended for many applications. The capability to scope queries (*scoping*) with certain limitations is also highly different between existing SD systems.

The need for compact data representation in large service discovery system is apparent. The constant transfer of hundreds of thousands of detailed service descriptions over the network will result in the unnecessary increase in the network traffic. The propagation of detailed service description in resource-constrains environments also does not make sense. There exist a number of *compression techniques* [23], [24] that can be used to reduce the amount of transfered data. However, these compression methods are often relatively slow and complex or simply they are not needed, as more resource and bandwidth efficient methods can be used. *Membership testing* techniques, for example *Bloom filters* [25], [26] can be used to detect and propagate a specific information over the network. Generally membership testing methods allow to determine the presence of the certain object on the certain set, by conducting an analysis of the special summary set that incorporates the information about objects contained in a number of different sets.

There exist some issues that influence the performance of service discovery systems. *Bootstrapping* is a concept that needs to be considered with any realistic SD systems. Other issues include load balancing, pollution, churn rate and free riders problems that are encountered in large distributed discovery systems, especially in peer-to-peer networks. Security in general is, of course, a hugely important topic. However, due to its very importance we feel that security issues in these systems would call for a separate in-depth review, and we have chosen to consider fundamental security issues as being out-of-scope for this review.

II. Architecture Taxonomy

As mentioned SD systems have different architectures. A great share of SD systems implement a *structured* architecture. By structured architecture we understand systems in which the information about services is fixed on certain nodes. Such systems are in general faster in discovering services than unstructured ones and have a predictable service search time. Structured systems are further classified into *decentralized* and *centralized* ones.

Centralized structured systems follow either a classical client/server architecture or at least have a clear distinction between pure clients and hosts that provide additional services such as service directories. In general, servers store information about the services other nodes provide. When a user wants to find a certain service, he sends requests to the server (often called a *directory*) that returns him an answer. The centralized structured systems could be further divided to systems that are *highly centralized* or *autoconfigure-centralized*. With former we mean systems in which there is only a limited number of central servers and such servers are manually preconfigured to function as administrative servers. Examples of such systems are DNS [27] and LDAP [28] servers. Typically a client also needs to know the address (name) of these servers to use them, or at least need to be provided such information through secondary mechanisms such as DHCP. With the *autoconfigure-centralized*, or *bootstrap-centralized* systems, we mean directory servers which can be setup automatically using zero-configuration mechanisms. Hence, although there is a centralized local structure, one does not need the administrator to setup such services, and the number of such servers can be higher and even adapt to optimize the discovery service offered. Moreover, it is typically easier to find these servers automatically, e.g. by flooding or multicasting queries into the local network. Such architecture is used by many classical SD systems such as as Jini [18] and SLP [7]. Centralized structured SD systems have typically the fastest search time, but have potentially a single point of failure and are vulnerable to Denial of Service (DoS) attacks [2].

In *decentralized structured* architectures the distribution of the information concerning services is tightly controlled by the SD system. There exist no servers *per se* and the information is distributed almost evenly between all participating nodes in a predefined manner. Overlays of this type commonly make use of Distributed Hash Tables [4]. With this mechanism nodes need to index or store data that they may not be interested in. A hashing algorithm is used to identify the node who stores the searched resource. Connections in the overlay are predetermined by the indexing. Decentralized P2P architectures can be extremely effective for searching resources using unique identifiers, but they fail when a search using partial matching is required. Additional overhead is created by managing the network architecture, updating the structure in the case of node

[2]Other structured telecommunications systems such as ISDN/SS7 [29] and nowadays SIP [30], of course, also carry out functions akin to service discovery. These solutions have proven their scalability extremely well, but tend to be very dedicated in their nature. Accordingly, we shall consider them as being out of the scope of our review.

failures or node arrivals [31], [32]. DHT-related methods are further described in Section VI.

Systems built as *unstructured overlays* support partial-match and complex queries. The use of loose architecture makes the network resistant to node failures and DoS attacks. Systems adapt easily to frequent node joins and disjoins. On the other hand unstructured P2P systems do not discover rare items as efficiently as the popular ones. Their performance depends heavily on the search mechanisms implemented. Usually systems employing pure unstructured architecture, for example Gnutella [6], do not scale well for very large networks. That is why the pure unstructured P2P frameworks have nowadays mostly been replaced by peer-to-peer overlays that utilize a hybrid architecture instead. Finally, *hybrid frameworks* try to combine the advantages offered by the above types of architectures. Most often they make use of clustering to introduce a loose structure to the unstructured network. Clustering is described in more detail in Section V.

III. Enabling techniques

In this section we consider some basic enabling techniques for service discovery, as well as introduce some additional parameters by which the discovery frameworks can be classified. We first discuss the scale of different SD systems, after which we consider different packet propagation methods and shortly describe alternatives for information spreading. Discussion of dynamic query termination mechanisms concludes the section.

A. Packet propagation

The careful choice of the packet propagation techniques for a specific SD protocol is very important. The choice can depend on a variety of parameters like the underlying network topology, communication media or the foreseen scale. For example multicasting is a temping technique in case of the local scale systems designed to work in the fixed networks. However, for wireless environment one might want to take the full advantage of the inherent broadcast nature of the wireless medium. For large scale systems unicast is a good option. Below we provide an overview of the four main packet propagation methods used in most networks, and especially ones based on IP: *unicast*, *broadcast*, *multicast* and *anycast*.

1) Unicast: Protocols that involve just one sender and one receiver, i.e. one-to-one communication, are referred to as *unicast* protocols. Unicasting can be employed for sending requests to dedicated nodes (servers) that possess information about services that other nodes provide. The technique is also employed when the node hosting the service is already found, but the information about the service has to be updated. Unicasting is often used in complex search protocols, e.g. the random walk and its derivatives.

2) Broadcast: Broadcast protocols use one-to-all communication, where the packet is sent to all nodes in the subnetwork or the whole network. Service discovery frameworks employ broadcasting to discover resources in the absence of servers. Broadcasting is often used to advertise a new service appearing in the system. The technique is widely employed in wireless networks, due to the broadcast nature of the communication media.

3) Multicast: Multicast protocols [33], [34] employ one-to-many communication that allows a sender to propagate a packet to the group of nodes with a single send operation. Classical multicasting is mainly used in fixed networks, as in wireless networks the full multicast support is quite costly. This packet propagation method requires a comparatively complex mechanism to be implemented on the node. The building and maintenance of the multicast group clearly imposes extra overhead on the network. For wireless networks with mobility the cost of classical multicast support can turn out to be just too high and it might be more efficient to use simple broadcasting.

Multicast is often used in fixed networks to discover services in the absence of a server, for example in SLP. The advantage of multicasting over broadcasting in such a case is clear, as the service request can be propagated only to the group of nodes that are more likely to answer the request. A good example is the propagation of a request to the users belonging to the same interest group. Multicasting is often employed in complex search protocols.

Sometimes under multicasting is understood the process of message sending to the specific multicast address. For example Bonjour does so. This propagation method is somewhat similar to broadcasting. All interested nodes are listening to the specific multicast address and process messages coming to it. However, multicast packets are just broadcasted through the subnetwork. This type of multicasting is applicable to small networks, both fixed and wireless. For the large wireless networks such multicasting might need modifications in order to provide better scalability. For example, the probabilistic broadcasting or gossiping [35], [36] can be used instead of simple broadcasting.

It is often said that the *publish/subscribe* [37] method is in nature similar to multicasting. In publish/subscribe a user subscribes to certain services provided by publishes, and obtains them either directly or through intermediate nodes. All changes in the services are reported to subscribers. In case if an intermediate node is present, the updates may not be propagated to the subscriber immediately. Instead they can be published by the source nodes on an intermediate node and cached there. When the subscriber has an opportunity it can contact the intermediate node and fetch the data.

4) Anycast: Anycast [38] is as multicast a one-to-many communication method. However, packets are delivered not to all anycast destination points, but only to one of them that best complies to the given parameters. Basically in anycast mode we have a point-to-point communication with the end point chosen from a pool of available destinations. The destination node can be determined using a variety of metrics such as closeness to the sender, minimal load, number of neighbors, etc. As with multicast, all nodes belonging to the same anycast group are listening to one anycast address. Due to the dynamic nature of anycast packet forwarding, this technique works better with connectionless protocols (such as UDP) rather then with connection-oriented protocols (TCP for example). Anycast can be used for various purposes, for example it is employed in DNS to determine the appropriate DNS root server [39]. In service discovery frameworks anycast can be used, for example, to achieve load balancing between cluster heads or nodes hosting the replicated data.

B. Reactive vs. proactive announcements. Pull vs. push

There exists two ways users can learn about services the network provides: *reactive* and *proactive*. When the sender explicitly requests to discover the services, it employs the *reactive* approach, i.e. the information about services is sent as a reply to the sender's direct request. Nodes learn about new services via *proactive* approach if they receive service advertisements without explicitly asking for them. Most SD systems use the reactive approach, for example Gnutella2 [10] and eDonkey [40] do so. Some, however, support proactive advertisements as well, Bonjour [22] being an example. Though it might seem that proactive advertisements might never be needed, they are useful in special cases. For example they are employed in local networks in combination with aggressive caching to reduce overhead or in highly mobile environments. The alternative terms used are *pull* for reactive announcements and *push* for proactive data propagation.

C. Other ways to deliver information

There exist additional techniques related to the propagation of the service discovery information. Below we consider four of them. *Caching* is mainly used to reduce the network overhead. *Regular updates* are needed to decrease the amount of stale information in the system. *Tracing* is employed to propagate data along specific network paths. *Piggybacking* uses existing traffic to spread service information.

1) Caching responses: Caching is an effective method to avoid excessive traffic in the network. Users or intermediate nodes can cache successive service replies in order not to repeat the search later. However, care must be taken in order to avoid too much outdated, stale information to be stored. There are several ways to do caching. In classical SD systems servers usually cache the data. In unstructured P2P networks the data is either stored only at the sender node or at intermediate nodes through which the answer to the query passes. Sometimes the information is shared, e.g. using Bloom filters, between n-hop neighbors. In structured P2P systems information is located at the preassigned nodes using DHT tables. In hybrid systems super-peers typically store information and answer on behalf of their leaf nodes.

2) Regular updates, "Hello" messages: Regular updates are vital for SD systems to ensure that the cached service information stays up-to-date[3]. It is important to choose the timer values correctly to keep a fragile balance between overhead imposed by update messages and the amount of stale information in the network. Typically the timer values are set separately for each system. Sometimes they can be also adjusted during the run time depending on the network conditions.

Network nodes can regularly issue *Hello*-messages to exchange information with their neighbors. *Hello* packets may contain parameters that nodes need to know about each other in order to make decision on packet forwarding. Usually these messages are propagated to one- or two- hop neighbors. The advantage of this method is in the regular updates that nodes receive. The disadvantage is numerous additional packets issued in the network that consume networks resources. Unchecked injection of packets into the network is especially dangerous in wireless environment, as, this can lead to the drastic increase in the packet collision rate.

3) Using tracing: Tracing messages are usually created at the nodes where the answer to the query was found. The messages are propagated via the paths the query traveled before. These packets are usually used either to spread information about the nodes they visited previously or to store data about the answer to the query. The advantage of this method is that usually fewer irrelevant messages are issued by the networks than with *Hello* messages, as the last ones may provide updates when it is not really necessary. The information transferred by the tracing messages is also likely to be propagated deeper into the network (not just two-hop distance), which might also be useful. The disadvantage, besides the additional traffic, is the lack of guarantee that the message will reach to the nodes that really need this data.

4) Piggybacking: In this approach additional information is added to the bypassing packets. This technique does not increase the number of packet in the network, though it will increase individual packet size. The problem is that the node can not control the destination of the used packet, so the information may not arrive to the interested nodes, or require several piggybackings to get there.

Organization of piggybacking requires careful consideration. There exist constraints on the maximum packet size used in different environments on physical layer. Therefore, if the packet on the upper layers exceeds maximum packet size on the physical layer, fragmentation will occur and the advantages suggested by piggybacking will diminish. Excessive piggybacking in extreme case can also lead to dropped packets, as for some protocols routers might not be able to fragment and further process packets which size exceeds the maximum transfer unit (MTU).

D. Dynamic query termination methods

One of the key issues in service discovery is the radius of the search. It becomes particularly crucial in the case of large diameter networks. The overhead of the search protocol depends heavily on the number of search messages issued, as well as on the hop distance each message travels on average. It is a delicate task to ensure that each message travels the minimum necessary number of hops to discover the resource and at the same time keeping the additional overhead to minimum. Below we present three dynamic query termination methods and one general modification that is applicable to all of them. We consider the performance of the termination methods with the respect to query propagation techniques of two major classes: flooding and random walk (Section IV-A), which are in turn based on multihop broadcasting and multihop unicasting.

[3]Under cached data we understand not only query replies, but also supplementary information, such as lists of neighbors or routing tables

1) Iterative deepening: Iterative deepening [41] is mostly used with flooding based search mechanisms. The random walk search mechanisms do not benefit from this termination technique. Iterative deepening uses successive floods with increasing values of TTL fields. Therefore, with each new iteration packets cover the larger network territory. However, this method imposes substantial overhead, by propagating through the large portion of the same nodes each new cycle. The TTL, i.e., the search diameter grows until either a searched item is found or the query timeout expires. Iterative deepening performs extremely well when popular, well-replicated items are to be discovered. For rare items standard flooding is more applicable.

2) Checking: Checking [41] is the termination mechanism used for random walk based search methods. Each query replica is propagated to randomly chosen neighbors of the nodes. The query is forwarded further until its TTL hop-count expires. At this point the query stops and a special checking packet is sent to the source node. This packet checks if the originating node has already received an answer to its query. If not, the checking packet is sent back to the node where the query is residing, carrying the command to carry out the search further. Otherwise, the search stops and the query is deleted after a certain timeout. The checks to the source node are usually done every n hops. This termination method is to be applied to search for popular resources, otherwise the basic random walk performs better [41].

3) Chasing wave: The chasing wave termination method [42] can be applied to both flooding and random walk techniques. However, it performs slightly better for the random walk, losing in performance to iterative deepening when combined with flooding. The functioning of the method is analogous to the chasing game, where the slower query replicas are chased by fast query termination packets. All query replicas are propagated through the network with increasing on-node delay as the distance to the source node becomes greater. At every node the packets leave marks indicating where they are traveling further. As soon as any query replica obtains an answer to the query it issues the chasing packet that is sent after all the query packets terminating them. To find the query replicas the chasing packs utilize the marks left by queries before [42].

The simulations carried on all of the termination methods have shown the advantages of the *adaptive techniques* used on top of the basic termination methods [42]. In most of the cases half of the overhead was removed. The suggested adaptive techniques make use of the search history. The initial TTL, as well as the TTL increase step, for each termination method is adjusted based on the hop distance at which the resource of the certain type was found during the previous search. This data is disseminated through the network using piggybacking, resulting in shorter search paths and therefore smaller overhead. The price for the use of history-based adaptive techniques is the increase in the amount of state information each node has to maintain.

IV. Fundamental search methods and protocols for unstructured networks

Different techniques can be used to discover resources on the network. If the network is small no complex search techniques are needed. One can use simple broadcasting or multicasting for querying. Centralized systems with few servers also do not require complex query propagation methods. However, if we want to support complex or free-form queries in decentralized networks, such as unstructured P2P overlays, sophisticated search techniques have to be applied to query propagation to achieve scalability and efficient operation. Below we give an overview of the most common search algorithms and example protocols that utilize these methods.

Search in a graph is defined as finding a path from a start node to a destination node. In our context the destination node is the node that contains the service searched. The *cost* of a search can be defined in various ways, for example as the number of edges traversed in locating the destination node or the number of packets sent into the network during the search process.

The search algorithms are classified into *uninformed* [43][4], where sending nodes know nothing of the surrounding networks, and *informed* [44] that rely on the partial network information discovered previously. Informed and probabilistic methods significantly reduce the overhead in the system, but they suffer from the partial coverage problem, in extreme cases showing very poor performance.

A. Uninformed search methods and related protocols. Search in unstructured networks

Uninformed search methods [43] can be further divided into *systematic* and *random* search algorithms. *Systematic search methods* typically explore the searched tree or graph according to some predefined rules. There is no place for probabilistic or random choice in such methods. Often systematic methods conduct a complete or almost complete search of the studied graph. The theoretical evaluation of the considered uninformed systematic search methods is given in Table I, where the notation established in [30] is used: b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are used as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} is optimal if step costs are all identical; ^{d} is optimal if both directions use breadth-first search.

1) Breadth-first search and flooding: The most well known search algorithms in this category are *Breadth-first (BFS)* and *Depth-first (DFS)* searches. BFS first explores all neighboring nodes of the sender and, if the solution is not found, proceeds to explore all the two-hop neighbors. The depth of the search is further increased until either a solution is found or all nodes of the network have been searched.

[4]We provide just two common references for the description of uninformed and informed search methods for the compactness of the survey. Each individual search algorithm can be found in the referred book chapters [43], [44] if not stated otherwise.

Criterion / Name	Breadth-first	Depth-first	Depth-limited	Iterative-deepening	Uniform cost
Complete?	Yes[a]	No	No	Yes[a]	Yes[a,b]
Time	$O\,(b^{d+1})$	$O\,(b^{m})$	$O\,(b^{l})$	$O\,(b^{d})$	$O\,(b^{1+\lfloor C^*/\varepsilon \rfloor})$
Space	$O\,(b^{d+1})$	$O\,(bm)$	$O\,(bl)$	$O\,(bd)$	$O\,(b^{1+\lfloor C^*/\varepsilon \rfloor})$
Optimal	Yes[c]	No	No	Yes[c]	Yes

TABLE I
EVALUATION OF UNINFORMED SYSTEMATIC SEARCH METHODS (ADAPTED FROM [43]).

Flooding [45], [46] is one of the basic search protocols. Its foundation lies on the BFS search algorithm. The query is propagated to all nodes in the network. The number of query packets typically increases exponentially further the query travels from the source node, causing huge overhead. The basic solution is to introduce a time-to-live field that limits the query propagation to a certain hop depth.

2) Depth-first search, related methods and protocols: Depth-first search explores one neighbor of the sender. If solution is not found it increases the depth of the search, i.e. explores one neighbor of the previously searched node. The search continues in depth: each new node searched is situated at the increased hop distance from the requester. If maximum depth is reached than the search path is traced back until it can branch and go again in depth.

Depth-limited search is a special case of the depth-first search, where only nodes with depth less than some bound are considered. Famous time-to-live (TTL) field is means for implementing this principle. *Iterative deepening depth-first* repeatedly applies the depth-limited search. In each iteration the maximum search depth is increased and the search is re-run. During the iteration BFS algorithm is applied.

Iterative Deepening [46] is based on the depth-limited search algorithm. It can be considered as a separate search technique as well as a packet propagation termination method. Iterative Deepening is further described in Section III-D.

3) Uniform-cost search: The search starts with a root node and all the neighboring nodes are explored. The neighbor connected to the root node with the lowest path cost link is chosen. Then all possible neighbors of the nodes, that already were chosen, are searched again and the node with the lowest path cost is preferred. The process continues until the searched object is found. Uniform cost is always *optimal* (since at any stage the cheapest solution is chosen). *Dijkstra's algorithm* is an example of uniform cost search.

4) Random search methods and protocols: These methods are governed by random variables. Good performance is expected, but not guaranteed by these algorithms. The *random walk* is one of popular random search methods [44]. This strategy is a formalization of the intuitive idea of taking successive steps, each in a random direction. A random walk is a simple stochastic process. The mathematical properties of random walks are quite well known for a long time, one of the celebrated examples being the Brownian motion.

Random Walk (RW) protocol implements the random walk search technique described above. It is one of the basic and most widely used networks protocols. Its properties are discussed in detail by Q. Lv *et al.* in [46]. The source node sends a query to a fixed number of neighboring nodes. The number of query replicas does not increase with the hop distance. The method allows in many cases considerably reduce the overhead imposed on the network with a tradeoff of the reduction in hit rate, increased round-trip time, and highly variable performance.

Probabilistic forwarding [47] is another of the random search methods. For each node where the query might be forwarded a random number in range [0, 1] is generated. If this number exceeds a threshold then the node is searched, otherwise it is skipped.

Probabilistic Flooding protocol [35] is based on flooding, but the query replicas are forwarded here only to a certain percentage (p) of the node's neighbors. Normal flooding is an extreme case of the probabilistic flooding with $p = 1$. If $p = 0$ then the query is, of course, not propagated anywhere. Probabilistic flooding is one of the often proposed techniques to be used in large dense wireless networks. There the propagation via standard flooding, besides leading to huge overhead, also increases the collision rate and leads to the degradation of the network performance. Careful choice, maybe even an adaptive adjustment, of the probability of further packet propagation leads to a drastic increase in the protocol efficiency and at the same time keeps high the probability of the packet delivery to the destination. Use of this method in sparse networks is not justified, as there the sufficient probability of the packet delivery can not be ensured.

Gossip-based protocols [36] are based on probabilistic flooding. A node forwards a messages to a certain number of its neighbors if it believes that they have not already received a certain amount of the message replicas. The protocol is especially suited to the large-scale distributed systems with limited mobility and high node failure level, i.e., when the connection between nodes rarely change, but the nodes themselves are often unavailable.

B. Informed search methods

Informed search methods [44] make extensive use of *heuristics*. A heuristic is a method that does not guarantee the solution found to be optimal, but usually finds an acceptably good solution in a reasonable time.

Best-first search goes through a list of possible nodes to explore and chooses the most promising ones to be explored first. Heuristic is used to rank the neighboring nodes based on the estimated cost from the current node to the solution. There are several variations of this algorithm. *Greedy search* algorithm chooses the node that appears to be closest to the goal node from the current node. The algorithm makes the locally optimal choice at each stage with the hope of finding the global optimum. *Beam search* is similar to the best-first search, however it unfolds not one, but the first m most promising nodes at each depth.

*A** search falls into the category of best-first searches. The algorithm takes into account both the cost from the root node to the current node (function *g(n)*) and estimates the path cost from the current node (function *h(n)*). Function $F(n) = g(n) + h(n)$ represents the path cost of the most efficient estimated path towards the goal and is continuously re-evaluated while the search runs in order to arrive at the minimal cost to the goal. A* is monotonic, it is complete and optimal on graphs that are locally finite and where the heuristics are admissible[5] and monotonic.

Adversarial search methods, e.g. MiniMax and Alpha-Beta Pruning, are rarely used in SD search protocols, so we will not describe them in detail here. The interested reader is referred to [48] for discussion.

Backtracking, falling in the class of *constraint searches*, is used to find solutions to problems specified by a set of constraint variables. Backtracking in the worst case tries all possible combinations in order to obtain a solution. The method's strength is that many implementations avoid trying many partial combinations, thus speeding up the running-time. The term "backtrack" was coined by American mathematician D. H. Lehmer in the 1950s [49].

Hill climbing falls in the class of *Iterative improvement methods*. The method extends the search path with a node which brings the path closer to the solution than it was before attaching the node. Two major modifications of the algorithm are used. In *simple hill climbing* the first node that brings the user closer to the solution is chosen. In *steepest ascent hill climbing* all possible nodes are compared and the closest to the solution successor is chosen. Finding of only local maximum (in the case that heuristic is not good enough) is the main problem with hill climbing. Several methods exist to overcome this drawback, including iterated hill climbing, stochastic hill climbing, random walks, and simulated annealing.

Lately, *simulated annealing*, a generic probabilistic meta-heuristic for global optimization problems, was tried successfully as a possible search algorithm for unstructured P2P networks [50], [51]. With this algorithm a fair approximation to the global optimum of a given function in a large search space can be achieved. Simulated annealing allows the system to move consistently toward *lower energy* states, yet still *jump out of local minima* due to the probabilistic acceptance of some upward moves during the first few iterations.

Tabu search is a local search algorithm that uses memory structures, tabu-lists, forbidding the use of certain values of attributes in the search. Tabu lists containing the prohibited values are very effective, though a very good solution that just happens to have this value might be missed. To overcome this problem *aspiration criteria* are introduced. They allow to override the tabu state of a solution and include the solution in the allowed set.

Ant colony optimization [52] is a meta-heuristic inspired by the behavior of ants in finding paths from the colony to food sources. It uses many ants (or agents) to traverse the solution space and find locally productive areas. The strategy usually does not perform as well as simulated annealing and other forms of local search, but it can solve tasks where no global or up-to-date perspective can be obtained, and therefore the other, in general more effective methods can not be applied. Ant colony optimization outperforms simulated annealing, tabu search and genetic algorithms in dynamic environments, as it can adapt continuously to the changes in real time.

Genetic algorithms [53] try to solve problems using techniques inspired by biological evolutionary mechanisms, such as yielding successive generations of possible solutions using reproduction, "survival of the fittest" and mutation methods. In genetic programming, the above approach is extended to algorithms, by regarding the algorithm itself as a "possible solution" to a problem. The genetic algorithm approach is used in *Immune Search* [54].

Neural search algorithms are based of the use of artificial neural networks (ANN) [55], that imitate the structure of the brain. Artificial neurons, modeling brain cells, are interconnected with each other to form a network using both forward and feedback links. The links connecting the neurons can have different weights, possibly changing during the run-time, thus producing an adaptive system. Based on these elements ANN creates a mapping between inputs and outputs, either deterministically or probabilistically. Neural search algorithms are widely used for speech and image recognition systems, pattern matching and search engines.

C. Protocols based on informed search algorithms

We now consider several search protocols that are based on informed algorithms. Most of them utilize the best-first approach. Many protocols are also using history-based metrics, to determine the destination or the next forwarding node.

Intelligent-BFS [35] and *Directed-BFS* [56] are informed versions of probabilistic flooding. The neighbors to which the query is to be forwarded are chosen judging from the success rate of the neighboring nodes for the queries of the same type. The answer packet, while traveling to the source node, updates the local indices on the bypassing nodes, increasing the probability that these nodes will forward these types of queries in the future. *Hybrid flooding* [57] is a further development of this approach. The algorithm uses probabilistic flooding to forward the query only to a subset of the current node's neighbors. Multiple weighted metrics are used to select these neighbors.

Degree-based random walk was first proposed by Adamic *et al.* in 2001 [58]. The algorithm is based on random walk and it issues a number of walkers (queries) that are forwarded to the highest-degree neighbors that have not seen the query. The neighbor connectivity is learned via the exchange of "Hello" messages. The algorithm shows a very good performance for power-law random networks when resources are concentrated in the high-degree nodes. However, if resources are not concentrated in the most-connected nodes, then the heuristic fails and can even perform worse than the basic random walk.

Distributed resource location [59] protocol makes nodes to listen listens to the bypass traffic and cache the information from the relevant search-related packets: the locations of the answer to the query as well as the description of the resource. Later, if the node receives a query searching for the

[5]A heuristic $h(t)$ is *admissible* if it is always less than the actual cost to reach the goal h^*, i.e. for every node in the network $h(t) \leq h^*(t)$. Admissible heuristic is optimistic, it never overestimates the cost to the goal.

resource the location of which is cached, then the query is forwarded directly to the relevant node, thus saving network resources and time. This technique is aimed for large and fairly static networks, where the initial discovery-caused overhead is compensated by saving in later requests. In a very dynamic network due to the outdating of the cached information the effectiveness of the method drastically decreases. The method exhibits increasing accuracy as the object popularity drops, as the less popular objects are less likely to be reallocated.

Adaptive probabilistic search (APS) [60] uses k random walkers and tracing messages to update supplementary information on nodes. The method employs distributed resource location mechanism. To choose the direction of packet forwarding a combination of probabilistic forwarding, historic learning and best-fist technique is used. The supplementary information on the nodes is updated using both positive and negative feedback. The nodes estimate using the history of previous requests the direction for the query to travel. However, a non-zero probability exists that a packet will be send to non-best fitting neighbor, thus enabling the protocol to explore new routes. A single entry is kept for each type of the resource for precise targeting. Compared to random walk, APS is rather a bandwidth efficient protocol and with additional adaptive techniques used [60] it achieves much higher hit rates. It is said that APS does suffer from partial coverage problem due to the use of random walk and informed propagation techniques. *Routing Indices* [61], *Query Routing Protocol* [62], [63] and *Local Indices* [45], [64] are examples of other protocols that extensively use metadata to exchange information between nodes.

Local Minima Search [65] is another search algorithm for unstructured networks based on greedy search and local minimum. It is somewhat similar to the mechanism used in DHT-based systems (Section VI) and suffers from the same inability to conduct complex searches. Each item is assigned a key, for example via hashing. Replicas of the *(key, value)* pair are propagated though the network looking for the local minimum between the key and the node ID. At the local minimum the replica is stored. The propagation method is a combination of random walk and greedy deterministic forwarding; this way the wide spread of the *(key, value)* pairs is ensured. The query for the item is propagated in the same fashion: first the key corresponding to the searched item is determined and then the search is started using random walk followed by the deterministic forwarding. Additional methods are employed by the protocol to improve its performance, such as dynamic adjustment of the number of replicas or use of bloom filters.

Immune search [54] method as well as *Genetic routing* [66] falls into the class of protocols based on genetic algorithms. The protocol consists of two parts: query propagation through the network and the topology evolution initiated as a result of search. The originating node issues a query, that is forwarded to its neighbors via random walk until the packet arrives to a node where similarity metric between the information profile and the message content exceeds the threshold. Then the message packet undergoes *proliferation* (more messages are issued) in order to be able to find more nodes with similar information in the neighborhood. Some of the proliferated packets are also *mutated*. Due to mutation the chance of message packets to meet similar items increases, which in turn helps in packet proliferation. Clustering (Section V) is introduced in the system to bring similar node together. The distance a node moves towards the query-originating node depends on the similarity between them, their distance and the number of times (age) the node moved before.

NeuroSearch [67] makes use of neural search algorithms (Section IV-B) and correspondingly neural networks to decide to which neighbor to forward the query. The decision whether to propagate a query to a certain neighbor is based on the output neuron of three layer perceptron neural network. Each neighbor is evaluated using seven parameters, such as was the neighbor connectivity or acknowledgment of the fact that a certain message was received before. Prior to the deployment the protocol needs to be trained on test networks to adjust neural network weights. A genetic algorithm is used during the training. With current array of input parameters the protocol performs well compared to flooding with low TTL on the network built after power-law distribution. The authors in [67] expect considerable performance improvement with introduction of history-based inputs.

D. Common exchange metrics

Nodes employing informed search methods exchange different types of information in order to predict the location of the searched resource. The examples of these metrics are a list of known services and their location, topology information, traffic load, power capacity, computational resources, communication channel quality, available bandwidth, historical feedback (e.g. number of successful queries forwarded) and node uptime. For the latter parameter it is generally assumed [68] that the longer the node stays without failure in the network the higher the chances are that it will be connected to the network in the future.

V. Clustering as a ways to construct hybrid systems

There exists a number of protocols that alter the network topology in order to increase the systems performance. Different approaches can be used, for example one can mix structured and unstructured networks architectures to enhance the search for both popular and rare content, or introduce peers that play a very specific role without interfering with the main service discovery functionality: search and publishing of the resources [69]. However, *clustering* is the most popular technique for creation of hybrid networks. The main aim of clustering is to keep such desirable properties of unstructured overlays as “loose” architecture, support of partial-marching queries and at the same time improve the scalability of the system. Several types of clustering are described below.

A. Super node clustering

In this clustering method *cluster heads* are elected dynamically among the nodes in the network. The alternative terms used to denote cluster heads are *super nodes, super peers* and *ultrapeers*. Cluster heads provide service information and conduct search on behalf of weaker nodes. The search process is divided into two parts. First the search among cluster heads is conducted, as they index the resources that a group of ordinary (leaf) nodes offers. Then, when the resource is located by one cluster head on another super node, then this super node typically forwards the query to the respective leaf node that hosts the needed resource. It is also possible that the cluster head does not forward the query to the leaf node, but directly sends the answer to the requesting node on the behalf of the leaf node.

Typically in order to elect a cluster head nodes exchange an array of parameters, e.g. processing power, throughput or number of neighbors, to choose the most powerful nodes to become cluster heads. The role of cluster head can pass from node to node while the network is running. The search among cluster head can theoretically follow any of the described search protocols, for example FastTrack presumably uses flooding [70].

The presence of cluster head nodes allows to drastically increase the scalability of the system. However, the maintenance of the additional overlay causes extra overhead, as information on the super peers have to be regularly updates by the leaf nodes. Therefore a careful choice is to be made regarding the cluster sizes and the data flow between ordinary and super nodes. The presence of super nodes also makes the network more vulnerable to the failure of specific nodes, as the simultaneous failure of several super nodes could cause a significant degradation in the network performance.

The problem of maintaining of the optimal super node/leaf node ratio without generating a large number of supplementary packets is addressed by *Dynamic Layer Management algorithm* described in [68]. To operate the method requires super nodes to possess information about all their leafs, and leafs to know a number of their recent super peers.

Super node clustering is quite popular and is used in such well known protocols as FastTrack [11] and Gnutella2 [10]. The respective protocols are described in Section VII. Other examples of the protocols and systems that use super node elections are Local Indices [56], Category Overlay [71] and Edutella [72].

Tree-based or hierarchical clustering is a logical extension of super node clustering. Here additional overlays on top of super node overlays are constructed to increase the scalability of the system. In such case the search is divided into the number of groups equal to the number of overlays. The approach suffers from the same problems as super node clustering, but on the bigger scale, namely failure of cluster heads and more complex support traffic.

In *locality clustering* nodes are clustered on the basis of the physical proximity. Physically close nodes, in terms of delay or number of hopes, are brought together to form a cluster. The super peer clustering described before is often indirectly based on locality clustering, as the cluster head is elected and responds on behalf of the group of nearby nodes. *Logical clustering* [62], [73] employs both locality and hierarchical clustering. The main idea of the algorithm is to ensure that all nodes can be searched within the given time frame. Therefore this algorithm may also be used as the basis for providing Quality-of-Service in overlays. A cluster in this method is defined as a subset of nodes that can be covered in a certain time frame by a single walker. The clusters elect cluster heads that communicate with each other using walkers as well. If the number of cluster heads grows so that they can not be covered by a single walker in a given time then a next hierarchical overlay is constructed from super nodes. At the end the logical clustering structure reminds a tree, where several levels of nodes exist that communicate with each other via walkers. Occasionally the messages are passed between different levels. The method shows a good performance in a fairly static environment, but fails in the dynamic one due to the lengthy hierarchy organization procedure.

B. Similarity and associative clustering. Semantic overlay networks

Similarity [74], *associative clustering* [75], and *semantic overlay network* [76] are terms that basically mean the same in the networking community. The new overlay is built on top of the original overlays on the basis of some rules that define the similarity between nodes. For example, for semantic overlays this is the similarity in the content stored; for associative clustering the similarity is defined as a node's interest in the particular area. Typically each node can belong to several similarity clusters depending on types of the objects it stores or is interested in. The approach relies on the assumption of high data correlation. If a node searches for a service belonging to one logical group it is very likely that he also will ask later for a service from the same group.

In associative clustering [75] a service request is at first propagated though a set of clusters using so called "guide rules", starting from the one with which the correlation is the largest, until an answer to the request is found. There exists a possibility for a node to form an individual search strategy in associative overlays based, for example, on previous search results. In comparison to unstructured overlays networks using these form of clustering typically show better results in locating rare items.

Interest-based locality [74] is quite similar to the associative clustering. This method creates an overlay of shortcuts that connect nodes interested in the same services. Shortcuts are updated of the basis of their performance, i.e. depending on how many queries get answered traveling via them. The shortcuts are ranked in a similar manner. The choice of which shortcut to follow the query makes is based on shortcut's rank.

W. Müller *et al.* [77] developed a distributed content-based image retrieval (CIBR) system for P2P networks. The CIBR systems require a high-dimension data to be used to describe and locate digital images. The authors propose a method for compact peer data representation by "cluster histograms", which also employ similarity clustering and elements of super-peer clustering in order to divide peers into interest groups with cluster heads that contain a summary of data of all "leaf" peers so that the search is first conducted within cluster heads and the ordinary peers join in on the later stage.

The *semantic overlays* is a widely studied area. In these overlays peers route queries to nodes who are interested in similar, in the semantic sense, information. For example, if a user searches for a Celtic song, the query will be routed also to the peers who possess any medieval music, as semantically these groups are close, and the peer interested in medieval music might also have or know another peer that has the desired song. The peers interested in similar information form a cluster, and the search for the certain content is at first limited to this area. Later, if the desired resource is not found the protocol might switch to an uninformed search. A. Crespo and H. Garcia-Molina bring semantically related nodes into groups called a Semantic Overlay Network (SON) [76]. The authors make use of a tree-like hierarchy to classify documents and queries and assign them to one or several SONs. Later the tree is traversed in order to locate the smallest possible range of SONs to search.

Large-scale content-based full-text search system called pSearch [78] constructs a semantic overlay using CAN DHT-based architecture (Section VI-C). Keys are document indices, semantic vectors formed using Latent Semantic Indexing [79]. Values correspond either to the documents themselves or the locations where they are stored. The authors claim that the system searches a very limited number of nodes and its performance is comparable to centralized (server-based) systems. GridVine system [80] also constructs a semantic overlay on top of DHT-based system, called P-Grid [81]. The authors emphasize that their system is scalable due to the use of a DHT-based network and is totally inter-operable due to the utilization of schema inheritance and Semantic Gossiping methods. The recent work by A. Löser *et al.* [82] studies semantic social overlay networks. Their routing algorithm performs routing based on the local neighbor information. The forwarding node is chosen based on the semantic closeness of the query to the content the neighbor stores. When the peer obtains enough data on previous search results the shortcut in the routing can be done, forwarding the queries to the desired peers without using information of the local neighbor. In the end semantically close peers form a list of shortcuts pointing to each other, thus, creating semantic overlay clusters.

VI. DHT-BASED SYSTEMS AS AN EXAMPLE OF DECENTRALIZED STRUCTURED ARCHITECTURES

Distributed hash table (DHT) based systems allow to find a location of a key's value given only the key, for example to find a location of the file given its filename [83]. The systems that implement this principle are fully decentralized, scalable, achieve load-balancing, provide a certain fault tolerance and most of them have theoretically provable correctness and performance. The main drawbacks of such systems is their inability to support complex queries, i.e. they support only exact-match search. Some works, for example [84], [85], [86], overcome this problem by introducing extra complexity to the system. This drawback comes from the fact that the main functionality that the DHT systems provide is the unique mapping of a key to the node that contains the key's value. Therefore keys and nodes should have unique IDs, that are typically generated via hashing. Another problem from which most of the basic P2P systems suffer is a low security level. The systems are vulnerable, for example, to Trojan and man-in-middle attacks [4].

The simplest DHT-based system operates like this. A user wants to publish a certain file in the network. The name of the file is hashed to produce unique file ID: the key. Then the function *lookup(key)* is called to estimate where the lookup *pair(key, value)*, in our case (file ID, file location)-pair will be stored. When the appropriate node is found, this pair is propagated in DHT overlay to the calculated node ID (Table II, Chord). If another user want to obtain this file the reverse procedure is initiated. First, he enters the file name, then via hashing its ID is obtained and fed into the *lookup(FileID)* function to get the ID of the node that stores the lookup pair. Via DHT-based routing the request for the File ID reaches the appropriate node ID and the file location is returned. Finally the user can download the needed file.

The main operation used in DHT-based systems is the *lookup(key)* function. It lies in the core of most other operations of the DHT-based system, such as a store operation (*put(key, value)*) or a retrieve value operation (*value=get(key)*).

The typical length of a key and a node ID is 160 bits. The keys are spread as evenly as possible among the nodes. DHT-based systems typically guarantee an upper bound in number of hops that a message will travel to achieve the destination node (Tables II, III). Hop advancement is large in the beginning of the routing and decreases with each hop traveled. Such strategy is achieved by utilizing a *distance function* that allows to calculate a logical distance between nodes. This function as well as the routing is special for each DHT-system and will be discusses later.

For the systems considered below we will not describe in detail how nodes join and leave the network. We will also not concentrate on the locality algorithms that they utilize or the replication strategies they use, because recent works in these fields (Section X) have addressed these issues providing solutions to some of the problems. We therefore describe only the underlying routing geometry [87] of the main DHT-based systems, as it, in the end, dictates the behavior of the system and limit its performance as a resource discovery mechanism. We also provide some basic characteristics of the considered systems (Tables II, III).

The variables used below are defined as follows: N is the number of nodes in the system, k is the key and B is the base used for node IDs.

A. Ring topology (Chord)

The routing space in Chord [88] is organized in a circle (Table II). The packets are always sent to one direction (e.g. clockwise) via the ring until they arrive to the destination node. For the system to function Chord requires for each node only the knowledge of its successor. However, to speed up the delivery process each node stores $O(\log N)$ elements in its routing table. In the routing table a node stores a pointer to the node half-ring away from it, one entry to the one-quarter node, one to the one-eighth node and so on. Therefore each node knows better its nearest neighborhood than far away nodes.

The (key, value)-pairs are placed at the first node, whose ID is equal to or greater to the value of the key. Chord

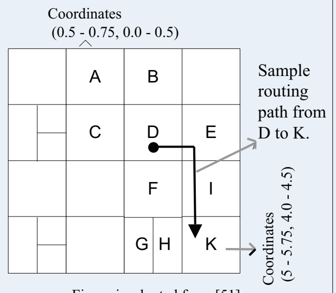
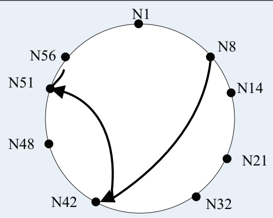
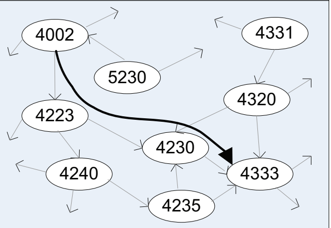
Name	CAN	Chord	Tapestry
Architecture	 Coordinates (0.5 - 0.75, 0.0 - 0.5) A B C D E F I G H K Sample routing path from D to K. Coordinates (5 - 5.75, 4.0 - 4.5) Figure is adopted from [51].	 The query for key N54 originates at node N8 and is routed to the node N56. Figure is adopted from [43].	 The query originates at node 5230 and is routed to node 4333 in Tapestry mesh. Figure is adopted from [48].
Lookup route length [Nr. of messages]	$O(d \cdot N^{1/d})$	$O(\log N)$	$O(\log_B N)$
Nodes join/leave [Nr. of messages]	2d	$\log N$	$\log_B N$
Number of entries/states per node	2d	$\log^2 N$	$\log_B N$

TABLE II
OVERVIEW OF DHT P2P OVERLAYS, PART 1 (ADAPTED FROM [4]).

uses consistent hashing [89] to ensure even spread of keys among the nodes. To make the system fault tolerant node besides maintaining a list of successors also has a pointer to its predecessor. The Chord runs a *stabilization protocol*, that periodically checks for consistency of immediate successor/predecessor pointers of nodes.

In Chord the lookup routing requires $O(\log N)$ messages and the routing table consists of the same amount of messages. The update of the systems in the case that a node joins or leaves results to $O(\log^2 N)$ messages (Table II).

There exist a number of other systems employing the ring topology. For example, F. Dabek *et al.* [90] have studied a variety of optimization techniques for DHT-based systems such as replication strategies, erasure coding, server selection, iterative and recursive routing, proximity routing and neighborhood selection. DHash++ optimized Chord-based system is a result of this study. Hybrid-Chord by P. Flocchini *et al.* [91] enhances the Chord performance and robustness by introducing some redundancy in the system via laying multiple chord rings on top of each other and using multiple successor lists of constant size. Chord-based DNS [92] and Cooperative mirroring/Cooperative File System [93] are other examples of the systems that use Chord.

B. Tree topology (Plaxton, Tapestry)

The tree-based routing algorithm (Table II) used by Tapestry [94] is based on the work of Plaxton *et al.* [95]. It uses correlation between a node ID and a key to route a message. The prefix in the node ID in the routing table entry is compared of that of the key. If their IDs share a common prefix at least in more than one digit compared to the current node ID then the message is forwarded to the considered node.

For a tree like search in Tapestry each node needs to maintain $\log_B N$ entries where B is typically equal to 4. This guarantees the delivery of a packet in $O(\log_B N)$ hops. The system's consistency is restored in case of a node join/leave in $\log_B N$ messages.

There exists a number of other DHT-based systems that utilize tree-like geometry. For example, K. Aberer *et al.* propose a self-organizing system called P-Grid [81] that is able to adapt to a changing distribution of data keys over the nodes. The system is further used in GridVine [81] semantic overlay.

C. Multi-dimensional Cartesian space topology (CAN)

The routing in the Content Addressable Network (CAN) [32] is done in a virtual multi-dimensional Cartesian coordinate space on a multi-torus; this structure is completely logical (Table II). Each node is assigned a unique area in the coordinate space. The coordinates of this area identify the node and are used in the routing. CAN uses greedy routing strategy where a message is routed to the neighbor of the current node that is situated closer to the required location. Therefore, for the effective routing a node needs to know only the coordinates of its neighbors and their corresponding IP addresses. Such routing strategy requires a continuous coordinate space without "empty", unassigned spaces. Thus when a node joins or leaves the system, the space needs to be dynamically reallocated, so that there are no "empty" spaces and each node has a certain zone to control. A new zone is usually obtained by splitting a zone of some random node into two parts. The absence of unallocated zones is ensured by enlarging of the zones of the nodes whose neighbor just left the network.

The allocation of the (key, value)-pairs is also done using the coordinate space. The pairs are mapped uniformly on the multi-torus using a hash function and each node hosts the pairs that were allocated to its zone.

The CAN algorithm is highly scalable, fault-tolerant and self organizing. Its lookup routing requires $O(d \cdot N^{1/d})$ messages and the routing table size does not exceed $2d$ entries, where d is the number of dimensions in CAN. The number of messages needed to react on a node joining or leaving the system is also $2d$.

Data replication rate can be increased in CAN by introducing a number of parallel coordinate spaces called *realities* so that each node participates in a number of realities, or by using several hash functions on the same space to obtain multiple coordinates for the same key. These strategies allow to increase the reliability of the system and reduce the average search query latency at the price of the systems complexity and resource consumption.

Due to its scalability CAN can be not only in traditional peer-to-peer applications, but also in large storage management systems like Farsite [96] or OceanStore [97].

D. Kademlia/XOR-tree topology

Kademlia [9] is a decentralized P2P network organized around *XOR* operation. Both peer addresses (Node IDs) and the keys are assigned values in the 160-bit space. The (key, value)-pairs are stored on the nodes, whose Node IDs are close to the keys in terms of XOR metric ($Distance = key \bigoplus textitNodeID$). To locate the key the Kademlia routing algorithm uses the same XOR metric to estimate the distance between the key and the Node ID of a peer. The node sends the query to those m nodes from its routing table that are closest to the desired key. The lookup process stops when the needed (key, value)-pair is retrieved. Additionally, the caching of this (key, value)-pair is done at the closest node to the key, that does not yet have the replica of the pair. The caching makes sense for Kademlia as it uses an unidirectional metric, which ensures that all lookups for the same key converge to the same path irrespective of the peer who issues the request. The caching, thus, allows to alleviate hot spots along the lookup path.

In the routing table each peer stores information (Node ID, UDP port, IP address) about other peers located at the distance from 2^i to 2^{i+1} from itself ($0 < i < 160$). To perform routing Kademlia maintains special lists, called k-buckets, where k is a system wide number, for example 20. Each k-bucket stores a list of nodes that are situated at the particular range of distances from the considered node. The distance is obtained by XOR operation conducted on the node IDs. different nodes are placed in one bucket if their distances form the source node correspond in the highest bit. For example the nodes that have distance metrics of "110" and "101" will be put in one bucket; the nodes with metrics "110" and "011" belong to different lists. Kademlia upgrades k-buckets when it receives messages from other nodes. This process is optimized to keep the longest living peers always in the routing table, as it has been shown that the longer the peer stays in the network the less probable it is that it will fail in the future [98].

Kademlia requires ($O(\log_B N)+c$) messages for the lookup routing, where c is a small constant. The routing table size is ($B \log_B N + B$) and number of update messages for nodes leave/join is ($\log_B N + c$) (Table III). The most well known examples of Kademlia usage are two of the overlay P2P networks for the eDonkey [40] client.

E. Viceroy/Butterfly topology

Viceroy [99] DHT-based overlay P2P network employs the butterfly network [100] in order to create a constant degree connection graph with logarithmic diameter approximation. Viceroy network uses identical to Chord mapping of the keywords to nodes, i.e. a key is mapped to the node with the succeeding ID on the Chord-type ring. However, the routing table contraction is different. A Viceroy peer maintains connections to its successors and predecessors on the ring for short distances. Additionally, each peer includes several outgoing links to chosen long range contacts. This long contact scheme is an approximation to the classical butterfly network. Each node belongs to a certain level l. There are in total $\log N$ levels in the network. The node makes two random connections with nodes of $(l + 1)$ level at the distance of $(2^l)^{-1}$ nodes away and two connections with nodes at $(l - 1)$ level. During the lookup routing the query is either sent to the levels $(l + 1)$ or $(l - 1)$, if the distance to the key is more than $(2^l)^{-1}$ nodes, or it is sent to the closest neighbors on the same level.

The constant out-degree network graph and some randomness in the link distribution allows to complete the lookup routing in $O(\log N)$ messages with nearly optimum congestion. The update in case of a node leaving/joining the network requires $O(\log N)$ messages. The routing table size is $\log N$ entries (Table III). Viceroy overlay network performance is proved formally in [99], however Kaashoek and Karger [101] point out the complexity of the algorithm and some fault-tolerant blind points in its construction. They propose their own DHT-based network that requires only $O(\frac{\log N}{\log(\log N)})$ messages per lookup request.

F. Pastry/Hybrid topology

The routing space in Pastry [102] is organized in a hybrid manner [87]. The search for a (key, value)-pair can be done either in a tree-like or, at a close proximity of a destination node, in a ring like manner. Ring approach is also used when the tree-routing fails. Both routing geometries were described previously in Sections VI-B and VI-A.

To maintain such hybrid geometry a node has to host three types of tables that assist the routing process. The size of a routing table used for tree-routing in Pastry is approximately $(\log_B N) \cdot (B - 1)$ entries. The leaf table that is used for the ring routing contains about 2^B and is used for ring routing. It contains half of numerically close larger node IDs and a half of smaller node IDs. Pastry will not fail until either half of the leaf table nodes fail simultaneously. Another table that each node maintains is the neighborhood table. It is used for locality routing, i.e. to route packets via physically proximate nodes. The size of the table is $2 \cdot 2^B$ entries.

Pastry has the following characteristics: the routing is done in $O(\log_B N)$ messages, the number of routing table entries per node is $2 \cdot B \cdot \log_B N$. The number of additional messages

Name	Viceroy	Pastry	Kademlia
Architecture	**Butterfly**	**Chord + Tapestry**	**XOR**
Lookup route length [Nr. of messages]	O(logN)	O($\log_B$N)	O($\log_B$N) + c, where c is a small constant
Nodes join/leave [Nr. of messages]	logN	2B·$\log_B$N	B·$\log_B$N + B
Number of entries/states per node	logN	$\log_B$N	$\log_B$N + c, where c is a small constant

TABLE III
OVERVIEW OF DHT P2P OVERLAYS, PART 2 (ADAPTED FROM [4]).

required for a node to join or to leave the network is $\log_B N$ (Table III).

VII. PEER-TO-PEER RESOURCE DISCOVERY

Peer-to-peer systems are responsible for the majority of traffic in today's Internet. Peer-to-peer applications include not only the classical sharing of files and other data, but applications such as messaging, media streaming and encyclopedia hosting. Service discovery is an integral part of any peer-to-peer framework. The success of a P2P system depends heavily on the service discovery mechanism involved, in particularly on its scalability and robustness. These overlay networks have created a whole new research direction in the service discovery that specializes on the development of SD schemes for large-scale P2P networks. In this section we give an overview of popular P2P systems, discussing their overall architecture, service discovery mechanisms they employ and describing features that distinguish these frameworks from others of their kind.

A. Napster

Napster [103] was the first commercial P2P system used to share mp3-files. The system was introduced in 1999 and at its peak had approximately 26.4 million of registered users [8]. Napster employs a simple client-server architecture, where the server indexes the resources its clients have. The basic communication is carries out as follows: a peer sends a search request to the central server, waits for a reply from it and downloads a searched file from the node to which the server pointed. Napster suffers from the disadvantages common for all centralized systems. Due to the legal issues the central server of original Napster was shut down and its era was over. Now Napster is again online, it employs the original architecture design and specializes in commercial distribution of mp3-files.

B. Gnutella and Gnutella2

Gnutella [6] was the second major P2P system that appeared after Napster. It was constructed to be decentralized and followed the classical concept of an unstructured P2P system. Bootstrapping of the network was done via any Gnutella node that was active in the network. The search for files was realized via the basic flooding. This method of searching suffers from numerous drawbacks (Section IV-A). Additionally, due to frequent peers disconnects, this network was never stable. The network stopped functioning satisfactorily as with such system structure the bandwidth cost of the search increases exponentially depending on the number of searched users. When the network grew large enough, it got saturated and often caused enormous delays. As the result, queries were often dropped and searches produced unsatisfactory results as only minor portion of peers were searched. Several improvements have been suggested to make Gnutella scalable, among them the use of degree-based random walk algorithm [58], introducing super-peer clustering (so called ultra-peers) [104] and different flow-control schemes [104], [105]. Unscalability of the original Gnutella inspired the development of the second version of the system, Gnutella2, that employs hybrid architecture with super-node clustering.

Gnutella2 [10] has two types of nodes: leaves and hubs. The latter are called super-peers or cluster-heads. Each leaf, normal peer, maintains one or two connections to hubs. Cluster-heads index resources of hundreds of peers by means of a Query Routing Table. The connected hubs also exchange hashes of keywords describing the resources that their leaves provide. During the search a peer sends a search request to a hub. If it is answered the peer downloads the file directly from the peer that hosts the resource. If the search is not successful, the request is forwarded by the current hub to another hub. Its address is taken from the routing tables of the super peers. The search stops when either the item is found or all known hubs are searched or a predefined search limit is reached. This approach considerably reduces the traffic in the network and makes the system much more scalable compared to original Gnutella. However, as a tradeoff, the complexity of Gnutella2 is higher than that of the original system and additional network maintenance required. The vulnerability of the systems to DoS and other malicious attacks on the cluster-heads increases.

C. KaZaA and FastTrack

KaZaA [21] and Morpheus [106] are the names of the different clients that connect to the *FastTrack* network, one of the biggest P2P file sharing networks today with over two million individuals online at any given moment [107]. FastTrack belongs to the 3rd generation of P2P networks. As FastTrack protocols are confidential, and FastTrack encrypts all network traffic, the system can not be directly studied and some facts that are presented below are results of indirect measurements and observations.

FastTrack network employs hybrid super-peer clustering architecture, i.e. two-level hierarchy to achieve necessary scalability. Ordinary nodes are connected to super-peers that index the resources the normal nodes possess and answer the search requests on their behalf. Therefore the search is conducted only among super-peers presumably via flooding [70]. Both types of peers are storing files. Super-peers are elected dynamically as the system operates. The parameters that influence the cluster head election are not completely known, but presumably nodes that possess high bandwidth, low latency, and plenty of computational and memory resources can become super-peers [108]. The work of Liang *et al.* [108] showed that one super node is typically connected to 40-50 of other super-nodes and supports about 50-160 ordinary nodes. At the time the work was conducted there were approximately three million nodes in KaZaA network, of which roughly 30.000 were super-nodes. Therefore the KaZaA super-node network is very sparsely connected, each super-node is connected to about 0.1% of other super-nodes in the network. The typical lifetime of a super-peer is around 2.5 hours.

The information that an ordinary node provides to a super-peer about a given file is a single filename, the file size, the file descriptors and the content hash that is used to identify the file in an HTTP download request. The choice to which super-peer to connect is taken based among other factors on the workload of the super-peer and the locality considerations. The list of super-peers available, as well as registration and logons, are provided to the user via a central server, which in principle can be a single point of failure in the system. The major disadvantage that KaZaa suffers from is a large amount of corrupt of false files that users can encounter during their searches.

D. eDonkey

The eDonkey [40], [109] P2P system is one of the most successful file sharing systems. eDonkey and its open source clients like eMule [16] or MLDonkey [110] have today more than 4 million users [107] connected online at any given time. The system is based on the eDonkey2000 protocol. The eDonkey network has a client/server architecture, where servers index resources their clients provide, but do not store any of the resources themselves. The largest servers index resources from approximately 900.000 users.

There exist three types of communications in eDonkey2000: server-server, client-server and client-client. Servers communicate over UDP to maintain the list of known servers. A client logins onto a server via TCP, by providing his username, IP address, the connection port and a list of files that are offered to the system. The server adds information about these files to the database and in turn assigns the user ID. eDonkey supports two kinds of user IDs: a High ID provides the full access the P2P network, a Low ID allows only restricted access. Low IDs usually have clients that are firewall protected or otherwise inaccessible. The server also sends to the user the list of other known servers. The list typically contains 100-200 entries. After this data exchange the user is free to search/download files, and meanwhile his files can be uploaded from the local machine. A user uses simple text search to locate files. This method allows to match the searched keywords with filenames and their descriptions. Substring searches are also possible. As soon as the searched file is located a user sends to a server "query sources message" that contains a hash of the desired file using MD4 Message Hashing algorithm. The server answers with ID/port pairs of the clients that claim to have a file. (In eDonkey use of hashes makes it possible to find files that have slightly different names, but the same content. Hashes are also used is to identify client nodes uniquely).

At this point client-to-client communication starts that is limited to the file transfer. A user downloads the file from all possible clients using multiple source downloading strategy. Such type of downloading is possible due to the fact that eDonkey separates files in multiple chunks, typically of 10 MB each, that can be downloaded separately. The complex scoring mechanism of eDonkey decides which upload request to serve next in order to maintain the maximal fairness between clients. To carry out this decision High ID/Low ID division is used.

Original eDonkey protocol has some legal vulnerability due to its use of servers. To overcome this drawback new decentralized P2P systems where developed for eDonkey and eMule [16] clients called Overnet and Kad. Both of these networks use the Kademlia DHT-based algorithm [9]. Nowadays these systems run in parallel with the original eDonkey network.

E. Freenet

Freenet [17], [111] belongs to the third generation of P2P networks that operates with various files. The main aim of the systems is to provide the maximum degree of privacy and autonomousy to the end users and protect their files from censorship. The network employs unstructured architecture. The request for privacy also leads to the redundant data replication, encoding of the information stored and to the great extend limits the end users from controling of the data allocated in the disc space of Freenet. The use of such a structure leads to slower and poorer performance of the system compared to other third generation P2P networks like KaZaA or eDonkey.

The system operates as follows: a globally unique identifier (GUID) is assigned to every file inserted into the system and is used later for file storage and search. GUIDs are calculated using SHA-1 secure hashes. Overall the system uses two types of keys. Content-hash keys are generated by hashing the content of the file and are used to generate GUIDs. Signed-subspace keys are used to set up a personal namespace that anyone can access, but only the owner can alter. This is done using public-private key pairs. In this way the anonymity of the user who uploaded the file is ensured.

The search for files is done using steepest-ascent hill climbing search (Section IV-B). The node forwards the query to its neighbor that it thinks is closest to the target. In case of a dead end or a loop the backtracking is used. The search stops when the query is either answered or the TTL value of the packet expires. When the file is found, it is replicated over the search path. This allows to cache more often the most popular files and protect the information from sudden nodes failures. Files are inserted in the same manner: a search is conducted for the file and if it is successful the file is positioned on all nodes that participated in the search. The files are deleted only if a certain node has run out of space. The deletion of files is performed in probabilistic manner, with less used files having a greater chance of being deleted than the popular ones.

Freenet employs key-based routing. The files are characterized by a set of keywords that are files themselves and therefore are also cached and have their own keys – GUIDs. A node forwards a query based one the closeness of the requested key to the ones that its neighbors are storing. The nodes create their routing tables based on the key information that they received from their neighbors. The keys do not rely on the content of the file, but just on the hash. Therefore files that have different content may still have close GUIDs. The system employs associative clustering, where files with close GUIDs are clustered together. This happens automatically during numerous searches and following replications, as files with close keys tend to be cached on the same nodes.

Freenet is a fine example of an unstructured P2P network. Though not so popular due to its performance issues its architecture suits well its original purpose. The scalability of Freenet has not been evaluated, but similar architectures have been shown to scale well in logarithmic manner [112].

F. BitTorrent

BitTorrent [113], [114] also belongs to the third generation of P2P systems. It is quite different from all of the systems considered above. First of all its focus is fast and fair download of files. The anonymity of users is sacrificed to a certain extent for this goal. Secondly, BitTorrent does not employ its own search algorithm, but makes use of central-directory based search facilities to host lists of files (with extension *.torrent*) that are available for download. Usually web sites serve for this purpose. A celebrated example is the `supernova.org` site.

To speed up the downloading process BitTorrent, as well as most modern P2P systems, employ multiple-source download mechanism. BitTorrent does it by splitting each file into smaller pieces and imposing a special bartering system on the users. The bartering system ensures that those who are interested in the particular file start to download different parts of it and then exchange them to complete the download. Such a system makes the download process relatively quick and efficient as both up/down streams of a TCP connections are used. Correspondingly the problem of "free riders" or "leeches" is also avoided[6].

To download a file the corresponding *.torrent* file should be found at the central-directory web site. Each *.torrent* file contains information about the file, its length, the hashing information with which the correctness of the received file pieces can be checked and the URL of the *tracker*. A tracker contains a list of peers that have parts of the searched file. This list does not contain all the possible peers that are interested in the file, but only some randomly chosen nodes. This is done in order to even the load on the network. Following the tracker the user finds other peers that are interested in the file and the exchange (download/upload) process begins. The general way for a user is to download the rarest pieces of the file first, leaving the popular pieces for later.

The BitTorrent network, though being very effective, suffers from legal issues as well as with difficulties to find and download old and unpopular files as the "swarm" of exchanging peers can not be created anymore.

G. Other P2P systems

We conclude this section by briefly describing some other existing P2P systems. *Manolito* P2P network [115] is based on Gnutella and, therefore, is completely decentralized. It uses UDP instead of TCP for search routing and full encryption to implement a higher level of user anonymity than original Gnutella. In order to connect to the network a user contacts a HTTP network gateway and obtains a list of nodes to which it can connect. A list contains approximately 100 entries. Depending on the available bandwidth the user connects to a number of them.

DirectConnect [116] utilizes a hybrid P2P architecture and is based on associative clustering, with each interest group being guided by a cluster head, a so-called hub. Hubs are pieces of software that organize the life of each cluster, but do not participate in the file exchange process. Hubs are located on central servers. DirectConnect is one of the most centralized hybrid P2P systems operational today. A user can freely choose his/her interest group/cluster and can directly exchange files or information with any other user in the same cluster.

WPNP (WinXP Peer Networking Protocol) [117] was a decentralized P2P network that was developed for WinMX client as soon as OpenNap community collapsed because of the legal issues. It used hybrid super-peer clustering architecture with a host cache server that served as a bootstrapping node. Though a remarkable P2P system in its time, it has experienced a decline during recent years due to not being able to compete with more young and innovative P2P systems such as eDonkey and Kazaa and was shut down in 2005 because of legal issues.

OpenNap [118] replicates the functionality of Napster peer-to-peer filesharing server. WinMX started to operate as OpenNap client. *Ares* [119] is a decentralized P2P network that has developed out of Gnutella and has a structure similar to

[6]Freeriders or leeches are the peers that only download from the P2P network without almost never uploading to it. Most of the current P2P systems suffer from this problem and try to fight against it, for example via introducing credits like in eMule [16] or enforcing bartering like in BitTorrent.

FastTrack. *Anatomic P2P* [120] employs a hybrid architecture with super-peer clustering and is based on *BitTorrent* protocol. The aim of the system is to decentralize BitTorrent network. *OpenFT* [121] was developed in the frame of giFT [122] project, it is a P2P system inspired by FastTrack. OpenFT employs tree-based clustering, using a thee-level hierarchy. Besides having user nodes and super-peers (search nodes) the system also employs index nodes to maintain indices of the super-peers. OpenFT is also remarkable for its use of "giFT: Internet File Transfer" [122] daemon that allows to dynamically load and use different P2P systems. Currently it supports Gnutella, OpneFT, Ares, FastTrack and OpenNap.

Currently, besides improving existing widespread P2P systems, there exists a trend to create clients that can support several P2P systems and switch dynamically between them depending on the user's needs. The examples are Shareaza [123], Morpheus [106] and the giFT plug-in [122]. Several works from academia, for example [124], [125], also address P2P overlay symbiosis and interoperability issue.

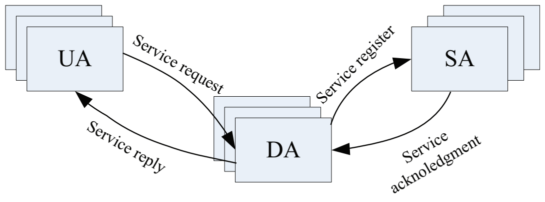
VIII. Dedicated Service Discovery Frameworks

In this section we describe the popular service discovery frameworks. In contrast to peer-to-peer networks that often concentrate only on the discovery of files, service discovery frameworks widely embrace the whole range of services that a local and enterprise networks can provide, for example CPU server usage, printing or obtaining sensor readings. Below we present six different systems. Service location protocol is an example of centralized enterprise-scale solution. The discovery framework for Bluetooth is a fine example a SD solution for resource constraint non IP-based networks. Jini is a popular Java-based SD system. JXTA is a decentralized hybrid Internet-scale platform. Finally, Bonjour and UPnP are examples of unstructured decentralized local-scale frameworks.

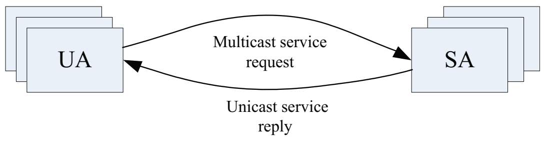
A. SLP

Service Location Protocol [7] (SLP) is one of the first well known service discovery frameworks. Though it was never as successful as UPnP, JTXA, Bonjour or Bluetooth, we consider that its short description is necessary as this framework is a classical example of a centralized SD system. SLP is aimed to provide service discovery in various networks ranging from LANs to large enterprise-sized networks. SLP supports two modes of functioning with directory agents and without them. The first approach is centralized and allows the service discovery framework to support a large number of nodes (Figure 2.a). The second approach does not scale as well as nodes simply exchange messages via multicast (Figure 2.b).

There are three basic roles that nodes in SLP can take: *user agents* (UA), *service agents* (SA) and optional *directory agents* (DA). One node can combine all these roles, for example be an UA and a SA simultaneously, i.e. both provide and use services. UA is, basically, a client that sends service discovery requests. SA plays the role of a server, providing services. A discovery agent collects service advertisements



A. Centralized approach. SLP with device agents.



B. Distributed approach. SLP without device agents.

Fig. 2. Two modes of functioning of SLP.

from service agents; user agents contact DA when searching for services instead of directly addressing service agents. User and service agents can discover a device agent either passively, trying to detect multicast advertisements, or actively issuing SLP requests. In case the central node, DA, is present in the network user agents contact it via unicast. Otherwise multicast is used by clients to query service agents. The protocol format stays in both cases the same.

Services in SLP are described via URLs using a special syntax scheme. The general format is `service:servicename:protocolname://hostname`. Additionally, each service can possess attributes and a type. A service advertisement includes the type, URL and attributes of the service. Service discovery queries contain either the searched service type or attribute specifications or a combination of the two. The service discovery reply contains the URL of the desired service. Additional mechanisms that SLP uses to enhance its performance, among others, include caching, authentication and scoping.

B. Jini

Jini framework [18], [126] is supported and promoted by Sun Microsystems. It is a platform and protocol independent system that relies on Java Virtual Machine (JVM) and uses the Java programming language to enable interactions between nodes in a network. Besides the Java support Jini also requires a presence of a reliable, stream-oriented transport protocol, such as TCP, and multicast support. The system can use Java Remote Method Invocation (Java-RMI) technique, somewhat similar to RPC, to conduct interactions between network nodes. Usage of Java-RMI allows Jini-enabled nodes to adapt to the network changes, as well as enables zero-configurability. Therefore this SD framework does not need, though allows, dedicated protocols to exchange discovery information.

Generally all services in Jini are represented as instances of Java objects and their descriptions can be seen as class definitions. Attributes of the services can also be described as separate objects. The service discovery architecture follows a central coordinator-based approach. For Jini the coordinator is named the *lookup service* and its functionalities are somewhat similar to ones of directory agent in SLP. The discovery process consists of two parts. First nodes detect and bootstrap to the lookup service in the network, after that they can conduct searches or publish their services. In the simplest case bootstrapping is done via multicasting. The lookup service is basically a repository of available services. Nodes can register new services, access, search and remove their services from the lookup service node. Service providers communicate with service users (clients) via special Java objects, called proxy objects, that are hosted as lookup service nodes. Jini allows creation of complex topologies as the lookup services can bootstrap each other, thus allowing clients to discover services that can not be obtained directly via multicast.

Jini is a very flexible service discovery framework, as it can be deployed in any type of network, provided that a Java Virtual Machine is supported there. It supports both RPC-based interactions between node, as well as a proprietary protocol-based approaches. The RPC-based interaction also enables zero-configurability of Jini-enabled devices, as these Java code chunks can perform any, not only discovery, function on the target nodes. On the other hand, the Jini architecture is vulnerable due to the presence of a central point of failure, which is the nodes that host the lookup service. The requirement for Java support also limits Jini from being installed on the range of resource-constrained devices, that do not have enough resources to support JVM. The workaround is to use a Jini-enabled gateway that can interact with the Jini network on behalf of weak nodes.

C. *Service discovery in Bluetooth*

Bluetooth is a widely spread short range wireless communication technology used for a variety of mobile and stationary devices, for example sensors, mobile phones, PDAs, printers, headsets and stationary computers [127], [128]. Bluetooth was first developed by Ericsson and now its technology specification is further enhanced and maintained by the Bluetooth Special Interest Group.

Bluetooth allows multiple devices to cooperate with each other and form small, private area sized, networks. The relations between devices are master-slave, i.e. each device plays the role of either a master or a slave. In one standard Bluetooth network (Piconet) there can be no more then one master device. Bluetooth is designed to function in the resource-constrained environments and to spend minimal amount of bandwidth and device resource[7].

Bluetooth is one of the few technologies that does not use IP-based addressing and employs different layer protocols to discover services. The service discovery operates on two protocol layers. The Link Manager Protocol, the Logical Link Control and the Adaptation Protocol operate on the lower level and allow to discover Bluetooth-enabled devices that can possess some services. The Service Discovery Protocol (SDP) operates on the higher layer and allows to discover the services on the devices.

[7]While one can argue about the optimality of the Bluetooth design, the fact is that it is the first successful technology of its kind and was widely accepted by the market. For this reason the service discovery part of the technology needs to be considered in our survey.

When the master node wants to perform the service discovery, it first sends a low-level inquiry message to all other nodes in range. All willing nodes respond to this message letting the master device know of their addresses. Additional query can be sent to retrieve the human-readable names of the devices. The full communication link is not necessary for that. After that the master device uses SDP to discover services that the other devices provide. The services are described via attributes. Each attribute in the (key, value)-pair form describes some parameter of the service like name, type, ID, etc. To save bandwidth SDP uses numerical 16-bit numerical tokens instead of string identifiers to denote keys in the (key, value)-pairs. As in many other SD protocols unique identifiers of 128-bits long are used in Bluetooth to denote some special values, like types or IDs of the service. The search query contains the list of the unique IDs (UUIDs) that correspond to the service searched or the attribute values of the needed service. The search for attributes of only the UUID type is supported; attributes the values of which are of other types can not be searched. SDP does not exercise partial-matching or complex queries to save the network and device resources.

Service discovery in Bluetooth is rather primitive, but suits its purpose. Many discovery mechanisms currently used in resource constrained environments were influenced by the Bluetooth solutions.

D. *UPnP*

The Universal Plug and Play (UPnP) framework [19], developed by Microsoft and UPnP Forum [129], defines a set of protocols that allow different types of devices to connect seamlessly to the network. A device can either offer or request services at any point of the network in a simple and standardized way. UPnP is built upon the standard Internet protocols, and it does not place any requirements on the operating system or programming language used to implement it. The service discovery model of UPnP is completely distributed query-based, which means it does not depend on a central server and thus is quite robust against communication errors. The standard defines two types of entities: *control points* and *controlled devices*. These correspond to Service Users (SUs) and Service Providers (SPs) when mapped to the general model of service discovery architectures.

Since the moment a device boots up, until it is ready to begin offering or using services, it goes through a series of steps defined by the UPnP standard, namely *addressing*, *discovery* and *description*. The UPnP Device Architecture (UDA) [130] allows two different types of addressing schemes to be used. First, a centralized addressing scheme (DHCP in this case) is executed; if this fails, a distributed addressing scheme (as defined in Dynamic Configuration of IPv4 Local-link Addresses [131]) is used. Name resolution is briefly commented in the UPnP specification. It is suggested that the DNS hierarchical name resolution could be used, although the document does not state that this is mandatory.

The service discovery in UPnP is based on the Simple Service Discovery Protocol (SSDP) [132]. The protocol allows controlled devices to offer their services, and control points to discover available devices in the network. In this protocol, controlled devices (those which host services) send unsolicited advertisements in the form of multicast `ssdp:alive` messages. All control points in range receive those messages and learn about the location of new services. The control points can also actively discover controlled devices in range by sending multicast `ssdp:discover` messages. Controlled devices may respond by issuing a message directed towards the originator of the request. In any case, control points receive a URL which points to an XML document describing the controlled device in detail. XML documents are the key to service descriptions in UPnP. An XML document named device description describes a controlled device in detail. This document can be retrieved from an HTTP server which runs on every controlled device. The device description contains pointers to other XML documents which further describe the device in detail.

Immediately after the description process is finished, the device is ready to begin using services. This is done by *control*, *eventing* or *presentation*. Control uses the Simple Object Access Protocol (SOAP) [133]. This is an Remote Procedure Call (RPC) mechanism which allows control points to invoke actions on remote devices and get the current values of the state variables. Eventing, on the other hand, allows control points to subscribe to controlled devices. This means, any time a state variable change occurs, the control point will be promptly informed. Eventing is based on Generic Event Notification Architecture (GENA) [134], which uses simple HTTP messages over TCP. Presentation is complementary to control and eventing, and offers an user-friendly alternative for using services. If a device has a URL for presentation, then the control point can retrieve a Web Page from this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and/or view device status.

The UPnP architecture is a work in progress. The UPnP Forum, which is a consortium of over 800 vendors, has set up working committees that are charged with creating proposed device standards, building sample implementations and appropriate test suites. Not only the UPnP Forum is working on improvements to the technology but also a number of researchers have found UPnP interesting, and have come up with proposals to modify the UDA. For example, Dabrowksy and Mills [135] propose an alternative to the M-Search mechanism for use in large networks. The M-Search mechanism is a part of the SSDP which extends the HTTP header with a new method, called M-SEARCH, to provide a basis for searching UPnP devices or services using a combination of HTTP, UDP and multicast. This new approach addresses the multicast query-response implosion problem which can occur in the case of a bad parameter selection. In addition a number of authors have questioned the choice of SOAP as a Remote Procedure Call (RPC) mechanism. Sanchez Santana [136] suggests an extension of UPnP in order to improve the performance of the system in wireless environment. Newmarch [137] proposes to break completely with SOAP, supported by the WorldWideWeb Consortium (W3C), and adopt Representational State Code (REST) as RPC mechanism. SOAP detractors argue their approach achieves more scalable, faster and simpler code. REST does not define extensions to HTTP methods, but only relies on URLs, XML and existing methods to invoke actions or query variables on remote devices. It is not a secret that UPnP suffers from scalability problems. This has its roots in the service discovery protocol used (SSDP), which is totally unstructured and lacks the possibility to use central directories to act as proxies for control points. The SSDP was conceived for SOHO networks where the number of devices will rarely be above 40 or 50.

Although UPnP is the most established tool for enabling simple and robust connectivity among consumer electronics, intelligent appliances and mobile devices from many different vendors, there is still a big room for improvements of the standard. The UPnP technology must evolve to adapt to the new challenges and trends set by the future networks.

E. Bonjour

Bonjour [22] is an open networking technology that provides zero-configuration capabilities, allowing different devices automatically form a network and seamlessly interact with each other. It is developed and actively promoted by Apple. The focus of the technology is mainly local and ad-hoc networks. Bonjour is used for a variety of applications such as file exchange, printing, music play list sharing and people discovery. The technology imposes no infrastructure on the network and can be implemented on a variety of IP-enabled devices ranging from mobiles to mainframes. Bonjour widely reuses existing technologies, such as Appletalk, multicast and DNS. The technology does not use any complex methods or algorithms and is made with the goal in maximal simplicity and error-proneness in mind. The system is designed for the cooperative environment. The closest competitor to Bonjour on the market is the UPnP technology.

Bonjour consists of two main components: automatic IP address acquisition and automatic discovery and use of the services on the network. The automatic IP configuration is achieved using the technique called link-local addressing [22] or DHCP service [138]. The service discovery in Bonjour relies on two main technologies: DNS Service Discovery (DNS-SD) [139] and Multicast DNS (mDNS) [140].

In the absence of a DHCP server the device tries to acquire a IP address in the local range, i.e. in the subnet 164.254.xxx.xxx. The IP address is chosen randomly by the node and it is sent to all other computers on the network. If nobody notifies the sender that this IP address is already in use the sender selects it, otherwise another random IP address is generated. The process is repeated until the unused IP address is found.

The device acquires the ability to discover services offered by other nodes in the network as well as to provide his own services via Multicast DNS-Service Discovery (mDNS-SD). The services of the new device are published and all network devices are notified of them via multicast. The whole syntax of the queries and messages are fully compliant with DNS standard, just instead of sending the message to a DNS server, the message is sent to the specific multicast address.

For the client the whole procedure looks exactly the same as convencial Internet host name resolution.

The service request might contain the name of service searched for, service type (the name of the protocol service uses) and the domain searched. For example when the search for the printer is conducted, the query will look like `_ipp._tcp._local`, where `_ipp._tcp` is the name of the protocol that device must support and `._local` is the domain name searched. By default the services on the local network are searched, i.e. the services with `._local` domain. However, it is possible to search any other domains on the network, like Google or one's home. The service reply contains the name of the service in human readable format, which is the unique service identifier in Bonjour, the service type, the domain and the additional attributes. For the printer example the answer can be `PrintingRoom._ipp._tcp._local` with the notice that the color printing is supported.

Bonjour's way to identify the services differs from many other SD frameworks. Most systems, for example JXTA or Bluetooth, use unique numerical identifies for that purpose, whereas Bonjour uses human-readable strings, which is justifiable on small sized networks.

Bonjour framework reduces the overhead by using aggressive caching and very infrequent service updates. The developers allow the cache to be stale, however when the mistake in the cache is discovered other clients also learn about it due to the multicasting nature of messages. Other overhead reduction methods used are suppression of duplicate queries and replies, exponential query and announcement backoff and service announcements on the start-up.

F. JXTA

JXTA is an open-source project, originally created by Sun Microsystems [141]. Here we describe the second version of JXTA. The aim of the project is to provide mechanisms that allow the creation of virtual overlay networks, mostly P2P networks, that will be platform and language independent and will function on the variety of devices ranging from mobile phones to mainframes. JXTA project is unique and successful in achieving this goal. Before it no such complex and comprehensive systems were built. The design of Jini [18], one of the first SD frameworks, is platform, but not language independent as it relies on Java Virtual Machine. SLP and UPnP [19] are language, but not platform independent, in a sense that for each new platform a new realization of the frameworks are to be written. JXTA relies on XML [142] for description of its resources, protocols and components, therefore any system that can parse XML can potentially use JXTA. Nowadays only the weakest, resource-constrained devices like sensor nodes lack this capability. Project JXTA aims to provides a variety of services over the virtual overlay network including the most popular ones like messaging and data sharing. This platform, besides providing a ready P2P architecture, also allows further research and development on top of its basic components, thus enabling developers to create systems optimized for their needs.

JXTA includes several standard components: advertisements, peers, peer groups, pipes, six basic protocols and strong security features. Below we provide short description of these components besides security, which is out of scope of the survey.

As other P2P systems JXTA uses unique identifies of 128-bits long. Unique IDs are assigned to every network resource, not just single peers like in standard P2P networks, for example in DHT-based networks. The term advertisements is also redefined in JXTA. Usually by advertisements we understand special types of messages that are send by a node to its neighbors to declare services it provides. In JXTA all network resources, for example peers, peer groups, pipes and services, are represented by special XML documents that are called *advertisements*. The description can be arbitrary complex and can be adjusted to carry a special kind of information, for example Web Service description with associated WSDL document.

Peers in JXTA are divided in two classes: *edge peers*, which are normal user peers, often characterized by unstable behavior or lack of resources, and *super peers*. Super peers are further divided into *relay peers*, that enable peers interaction over firewalls and NAT boxes, and *rendezvous peers*. The latter act as coordinators and store advertisements of the dependent edge peers. Rendezvous peers are almost classical super peers and create a further overlay on the basic JXTA network. Such construction allows to reduce the overhead imposed on the network and increase its scalability. The search for a resource in JXTA is conducted among super peers using a loosely-consistent DHT mechanism with a combination of the classical random walk. This approach can converge to classical synchronized DHT with optimum performance in case if the churn rate of rendezvous peers grows low and the overhead imposed by consistent index updates becomes justifiable. When the resource index is found the rendezvous peer forwards the request to the relevant edge-peer. Additionally, JXTA provides possibilities to employ more sophisticated discovery and search mechanisms if a developer wishes.

The scoping mechanism is provided in JXTA by using peer groups. We can say that JXTA uses a form of interest clustering. Each node can belong to several peer groups and the search queries will be propagated only in the relevant groups thus reducing network overhead. Peer groups can also be used to create secure domains or enable monitoring of the specific group of peers.

Additionally the system provides virtual communication channels, called *pipes*, to send messages and data. Pipes are asynchronous, unreliable and unidirectional. The pipes offer two modes of communication: point-to-point and propagation. The propagation pipe defines a connection that has one output and several inputs. The end points of the pipes can be replaced in case of peer failure without the pipe break. The core pipe types can be further extended by the developer according to his needs, for example to create a secure bi-directional reliable connection.

The JXTA includes six basic XML-based protocols. The Endpoint Routing Protocol is used to discover and manage a route between two end points. The Peer Resolver Protocol enables a peer to conduct the search within peers, peer groups, pipes and other network resources using generic queries.

The Rendezvous Protocol allows peers to subscribe to the propagation service that rendezvous peers provide. The Peer Discovery Protocol is used to find other peers, group of peers or advertisements in the networks. Peer Information Protocol enables peers to get status and capabilities of other participating nodes. Peer Binding Protocol allows to establish pipe between two or more nodes. Peers can function at a reduced level in JXTA if they do not support all of the above protocols. This leads to potentially thinner clients that can embrace JXTA.

The JXTA has been sometimes criticized being a rather complex framework. This argument definitely has some weight, as the JXTA system due to its generality has grown to be a complex and rich set of functionalities. As such it might not be the best suited for small-scale solutions, but as a general framework it is manifesting many of the crucial challenges and functionalities required from service discovery architectures.

Summing up, positioning of JXTA as an open platform and language independent modifiable project with strong security mechanisms contributes to the system's success. The scalability of the overlay network is achieve by using super peers and interest based clustering, that is powered up by DHT-based and random-walk search techniques. Numerous research projects exist on the base of JXTA. Among them are enabling JXTA on mobiles nodes [143], sensor networks [144] and grid computing [145].

IX. Challenges for service discovery in wireless and hybrid networks

This section is devoted to discussing issues specific to service and resource discovery in hybrid and wireless networks. We describe methods and techniques that are used for SD in wireless ad-hoc, mesh and resource-constrained networks. Some of the additional challenges that service discovery faces in heterogeneous networks are also outlined.

A. Ad-hoc and mesh wireless networks

Service discovery in *wireless ad-hoc (MANET)* and *mesh networks (WMN)* is particularly challenging due to their dynamic and multi-hop nature. Additionally, ad-hoc networks are infrastructure-less. The size of these networks typically does not exceed LAN scale, so scalability requirements are typically not very stringent. The main difference in the wireless environment compared to the traditional fixed one lies in the nature of the communication media: broadcast vs. traditional unicast. Broadcast environment favors limited number of transmissions as this decreases the number of potential packet collisions and therefore increases the network throughput. Typical clients for wireless networks are mobile phones, notebooks and PDAs. These types of devices are typically mobile and suffer from lack the power resources and, compared to ordinary PCs, lack of computing power. Accordingly SD systems in multihop wireless systems should not rely on expensive techniques (in term of bandwidth and power) such as classical multicasting or frequent flooding[8].

In wireless ad-hoc environment with high mobility it is impractical to build any centralized SD systems due to absence of stable nodes. Moreover because of constrained power resources no node can bear the role of the server for a long time. Therefore, only decentralized systems are feasible. In case of fast mobility the choice is further limited to unstructured SD platforms.

For wireless mesh networks there are not so many restrictions. The presence of fixed mesh routers with constant power supply allows the implementation of any type of service discovery architecture. Though there are few service discovery frameworks particularly designed for WMNs, we consider the adapted version of approach suggested in [146] to be suitable as the virtual backbone that the authors have suggested can be directly mapped on to the mesh routers. Generally we expect that cluster-based service discovery is particularly suitable for WMNs, as mesh routers are the natural cluster heads for surrounding mesh clients[9].

For wireless networks a new group of SD frameworks and protocols has appeared, due to the need to save number of transmissions and power. These frameworks transfer the SD functionalities from the application or overlay OSI layer to the network level, often incorporating service discovery and routing functionalities in one protocol. Such approach allows to limit the number of additional messages send and well as perform more precise routing for SD. Sometimes the resource discovery information is also piggybacked to the route discovery messages. One example of such an approach is the routing protocol called Virtual Ring Routing [147]. This protocol adapts the DHT-based routing for the network layer and introduces few modifications to take advantage of the broadcast environment. The protocol performs well in a variety of wireless environments, including ad-hoc, mesh and sensor networks. The structure of the protocol allow its easy extension to accommodate SD functionality in the spirit of original Pastry, i.e. keyword-based search. Other examples of protocols that bring the SD functionality to the network level are [148], [149]. Frameworks that follow a traditional approach in wireless environment and place service discovery on the application layer employ a variety of techniques, among them *clustering* [150], [151]; *push* [152]; combination of *pull* and *push* [14]; *decentralized* [153] and *zero-configurable* [22] systems.

B. Service discovery in resource-constrained environment

Next we consider service discovery systems for wireless resource-constrained environments that are characterized by the presence of low data rate communication media and highly resource and power limited devices. The typical example of such devises are wireless sensor network nodes, often called

[8]Though, in some cases, the use of these techniques is justified. For example, the first technique is useful in case of very low mobility and big network size; flooding on the contrarily is efficient in case of fast mobility and small networks.

[9]If wireless mesh networks are implemented on WLAN devices then the functionality of mesh routers are often performed by intelligent access points and WLAN mobile nodes act as mesh clients.

motes following the pioneering efforts at the University of California at Berkeley. The motes communicate over 802.15.4 radio and typically have few kilobytes of RAM and less than 100 KB of ROM. One mote should operate without a recharge at least half a year and in selected applications substantially longer.

Wireless resource-constrained (sensor) domain imposes even harder requirements on service discovery systems than the wireless ad-hoc environment. Low data rate communication media and the highly limited power resources do not permit high data flow and the use of sophisticated ARQ methods that almost guarantee the packet delivery. The typical packet size in such networks does not exceed 128 bytes. The limited computation resources stop the programmers from installing computational and memory hungry protocols and programs on these nodes. The same reasons do not allow the direct use of the most well known SD protocols like UPnP or SLP as they are string-based and their packets are simply too large to be processed by a sensor network. To overcome this problem one can either *simplify the needed protocol*, sacrificing some of its features to decrease the resource demands, as is done for SLP in [154], or use various *compression methods* [24]. However, not all of the compressing schemes are applicable due to the computational complexity. Nowadays the binary compression methods are popular, for example WBXML [155] is widely used to represent XML documents.

Another tendency in service discovery in resource constrained networks is the *merging of service discovery and data collection functionalities*. This happens due to the desire for resource economy and is also due to the fact that the vast majority of services (for example sensors) provide a limited number of different reading types, so that they can all be packeted to a single message. Examples of such approach are the TinyDB system [156] and SwissQM [13], the virtual machine designed for data gathering and processing.

Several mechanisms can be used to enable service discovery in sensor networks. The first one to mention is *cross-layer optimization* [157], the concept adopted from wireless ad-hoc networks [158]. Under cross-layer optimization we understand the breaking of the OSI layer hierarchy, when the nodes in the network can skip some of the OSI layers and the information exchange is enabled between any of the layers. Another approach is to use a different naming scheme for sensor networks based not on the topological location (such as IP addresses), but relevant to the applications that run on such nodes, so called *attribute-based naming scheme* [159], [160]. Attribute-based naming and cross-layer optimization allow to bring service discovery capabilities lower in the OSI stack, thus boosting the overall performance of sensor networks. Another approach that is proposed widely is *data aggregation* and *in-network processing* where some of the data processing and aggregation is done on the intermediate nodes, thus saving the bandwidth and power resources of the network [161], [162].

Two basic architecture models, client-server and distributed, find their place for service discovery in resource-constrained networks. The client-server approach expects heterogeneity in the network, where the more powerful devices can act as servers and carry the most of the computational load. Very often such devices also act as gateways to external networks. The architecture suggested by Schramm *et al.* [163] allows to integrate home-sized sensor networks into the SD architectures like Jini or UPnP. The list of SD architectures supported can be increased if more wrappers will be realized on the gateways. In this approach each gateway acts on behalf of the surrounding nodes, collecting data from them and generally controlling their behavior. The integration of new nodes into the network is thus handled by the gateway. The communication between gateways and ordinary nodes is done via a specially developed protocol optimized for resource-constrained environment that has nothing to do with external SD architectures in which the sensor network might be incorporated. These kinds of approaches have been proposed by a number of authors who suggest ways to integrate sensor networks into the IP world, see, for example, [164] and [165].

The distributed, P2P-like, approach tries to put minimal extra functionality on the gateways and treat all nodes equally. This approach is generally more scalable and is especially suitable for scenarios with random nodes placement in the large areas, such as environmental monitoring. Some of the designs that utilize these approach along with a number of the above-mentioned techniques (mainly data aggregation and in-network processing) are Directed Diffusion [159], Cougar [166] and Connected Sensor Cover [167].

C. Heterogeneous networks

Heterogeneous networks are composed of various network types, that in turn include different types of devices. The design of the service discovery system for a such diverse environment is not a trivial task. To our knowledge no universal solution exists and we believe that it might not even be possible to create a universal system like that. Instead, a loosely coupled solution in the spirit of Web Services [168] that allows easy adaptation to different types of networks and protocol stacks, might be a good option. Another possibility is to use solutions optimal to each network type and then have a integration service running on gateways that will allow to process and combine results obtained by different service discovery protocols. In some sense it is the same hierarchical, cluster-based, solution that is used to P2P networks, applied to complex large-scale heterogeneous networks.

X. Common challenges and complications

The design of a service discovery system is not a trivial task. One needs to deal with several issues in addition to basic search and service publishing. Among them are bootstrapping, security and privacy, search result ranking, handling of the node departure and failure, efficient resource management, data representation and load balancing. Large-scale decentralized systems additionally suffer from such problems as topology mismatching, pollution, collaboration issues, free riders and flash crowds. In this section we briefly discuss some of these issues in order to give the reader an overview of some of the possible complications.

A. Role of network topology in service discovery

It is intuitively clear that in extreme cases network topology can have great influence on the performance of a search mechanism or protocol. To see this simply compare a linear topology to a fully meshed network with the same number of nodes. However, it is far less clear how much differences between "typical" network topologies impact search performance. We argue that this impact can still be substantial, and care should be applied when reading about or carrying out performance evaluation of a new discovery protocol.

The broad classes of models for fixed network topologies usually considered are *random graphs* [169], *small worlds* [170] and *scale-free networks* [171]. Random graphs are the simplest of these models, and have the longest ancestry. They are formed by selecting a fixed number of nodes, and connecting each pair of nodes with a fixed probability. Small world models enhance this simple approach by including local clustering or correlations between connections between the nodes, and, finally, scale-free models include the observed power-law distribution of node degrees.

Perhaps the most influential work on relation between topology and search is that of Kleinberg [172], [173], [174]. He has, for example, studied several different network models and found that there are very large differences in the performances of search algorithms based on local information only in different topologies. Several groups have since studied the influence of, for example, the power-law exponent of a scale-free network to the spreading rate of queries. For an overview of these results see, for example, [58], [175]. Unfortunately much of this work has not yet found practical application, either in improving analytical performance estimates for existing protocols beyond the usual worst-case figures cited in the papers, or in actual topology shaping. Especially in overlay networks it would be perfectly feasible to tune the network topology to optimize search performance in addition to usual topological objectives, such as network reliability.

We conclude by few cautionary remarks related to wireless networks. It is again intuitively obvious that the models listed above should be poor matches for link-layer connectivity of a wireless network. The basic approach adopted in the literature has been to scatter nodes uniformly into an area, and connect close enough nodes by links. This yields a simple example of a *random geometric graph* [176]. While certainly already an improved model, much more could be done in this front as well (see [177] for some of the first work in this direction). For example, alternative clustered location distributions for the nodes could be considered, especially in the light of the recent experimental indications that real-world wireless networks might be strongly clustered [178], [179].

B. Overlay topology construction and topology mismatching

In theory most of the DHT-based systems can guarantee that any object in the network will be located in $O(\log N)$ steps. However, this number can not be directly mapped to the expected round-trip-time (RTT) value, as the overlay topology often does not match the underlying physical topology and we encounter so call topology mismatching problem. DHT-based systems use different mechanisms to route the queries via the faster and closer links. Several papers [95], [180] address this issue for DHT-based networks.

On the other hand it is not always useful to maintain a close correspondence between the physical and overlay networks. Quite often the overlay network has to be constructed to match a particular topology, for example a random graph topology. It has the most desirable characteristics for distributed unstructured systems: even work load, small diameter, resilience to node failures and high level of churn [181], [46]. However, real unstructured P2P networks rarely have this topology [182]. A membership management system CYCLON [183] allows to construct overlays resembling random graphs at the price of some overhead. The framework makes use of a gossip-based membership management protocol for peers to exchange and adapt their topology information. The work by M. Jelasity *et al.* [184] explores how dynamic unstructured overlays can be constructed and maintained via peer sampling using gossip-based approach and introduces a general framework to implement reliable and efficient peer sampling services.

W. Terpstra *et al.* used a random overlay topology and created a novel approach for searching in unstructured peer-to-peer networks, called BuddleStrom [185]. Their framework can handle arbitrarily complex queries as its network overlay structure is not dependent on the items reached, in contract to DHT-based systems. BubbleStrom relies on probabilistic search that is based on random overlay multigraphs. Both queries and data are replicated along the overlay of a certain diameter, creating bubbles. The replicas are not flooded inside of the bubble; rather, the information is distributed with a changing probability that ensures scalability and maintenance of load balance in presence of heterogeneous bandwidths. Intersection of query and data bubbles results in query resolution. Authors claim that their framework provides a reliable strategy for performing exhaustive search. Careful choice of the bubble size, evaluation of global system state, congestion control and replica maintenance allows BubbleStrom to be resilient to node failures and crashes, up to 90% and 50% of all peers respectively. The system was evaluated on a simulated 1 million node topology and proved to be scalable, fault-resilient and robust.

C. Bootstrapping

Under *bootstrapping* we understand the process that occurs when a node joins a network, in our case a service discovery network. There exist different ways to perform bootstrapping. In the simplest case, for a centralized system, a node can get the address of the server using DNS service. For the small network a simple flooding can be conducted to announce the presence of the node in the network. For larger decentralized networks another approach can be used. New nodes can join an existing system by asking a bootstrapping node for a list of nodes to which it tries later to connect. A central registry server or any node that already belongs to the network can act as a bootstrapping node. In the first case the central registry server is a single point of failure in the system. In the second a non-bootstrapping specific node may not have sufficient network information to assist the newly coming node in finding its correct place in the network.

After the node joins the system it learns addresses of additional nodes, for example via searching. Some time later this node leaves the system, still remembering the learned addresses. When the peer joins the systems again it will try to form links to all the nodes it knows about or at least with some randomly chosen ones. Such bootstrapping approach causes redundant links to be created in the overlay that will result in extra traffic in the underlying physical network. Another problem that arises is the mismatching between physical and logical overlay network links [186]. For example it can happen that one node is connected to another node via one logical link that actually corresponds many-hop slow link in physical network, instead of using the alternative of being connected by much shorter and faster link that exists in the physical network but not in the overlay network (or it is represented there by a several hop link).

There exist a number of alternative approaches that try to solve the above problems. End System Multicast (Narada protocol) suggested by Chu and *et al.* [187] uses the shortest path spanning tree constructed on top of network corresponding mesh graph. The root of the tree is the newly joining node. The overlay acquires new link in correspondence to the shortest path spanning tree. The overhead of this method grows proportionally with the size of the multicast tree and therefore does not scale well. The approach does not suit for dynamic environments. The Adaptive Connection Establishment (ACE) method [188] further develops End System Multicast, by constructing spanning trees of a certain network diameter and later establishing the connections between these trees. ACE is further improved by use of a location-aware topology matching (LTM) [189]. The algorithm allows to record a relative delay information and based on this data delete redundant or inefficient logical links. Liu *et al.* [190] suggested to use a coloring algorithm to construct links between peers, the systems is called scalable bipartite overlay (SBO).

The comparison of the above-mentioned methods [186] shows that ACE is the simplest of the methods and has relatively low convergence speed. The approach suggested by Liu [190] converges faster at the expense of increase in complexity. LTM allows to obtain the fastest results, but requires peers synchronization, for example using the Network Time Protocol [191].

Another approach involves locality clustering, basically linking nodes that are either close in terms of IP-addresses [192] or in term of the latency [193], [180]. To be precise the latency is measured not between two nodes, but between each node and a special node that support “landmark” functionality. The approach has the weakness of potentially constructing an unconnected overlay.

D. Support for more sophisticated queries

Standard decentralized discovery systems support either a limited keywords search (unstructured P2P systems) or index-based search (structured P2P frameworks). There exist a number of attempts to adapt distributed service discovery systems to support more complex discovery queries and enable search in large distributed systems, for example in distributed document collections [85], [194], [195]. These systems typically try to extend the search to support for relational and range queries, as well as to enable support for approximate answers. There exist two main strategies to enable complex and scalable query processing.

The most straightforward approach is to split the query propagation and processing functionalities [185] so that the query propagation direction is independent of its content and the query can be be processed independently using any existing external tool. This approach, though tempting, does not allow to use query content specific heuristic and thus limits the range of informed search techniques (see Section IV-B) that can be used.

The alternative is to keep the query propagation and processing functionalities coupled. However, in this case the careful balance between the query size, the request processing speed, the device resources and consumed bandwidth is to be maintained. J. Li *et al.* [196] in their feasibility study of peer-to-peer web indexing and search, have suggested two types of compromises that can be made in order to build a fast and scalable decentralized SD network. *A compromise in quality* of results that is done by ranking and transmitting only a part of the obtained results to the requesting node. *A compromise in structure* leads to the use of an aggregate bandwidth, local node resources, extensive use of caching and compression methods like Bloom filters, gap compression [197], adaptive set intersection [198] and clustering.

E. Other issues typical for peer-to-peer systems

Some issues in service discovery are only typical for specific decentralized systems, like classical peer-to-peer overlays. Nodes in these systems are free to behave as they want, however they carry a part of the network functionality and their failure or misbehavior can be rather costly. Below we describe some of the issues that peer-to-peer networks can encounter.

1) Fault tolerance and performance under churn: Peer-to-peer systems are characterized by the high instability of their nodes, i.e., by a high level of *churn*. By *the level of churn* we understand a portion of nodes in a network that have a short lifetime and have a tendency for frequent and unpredictable failures. An unpredictable node failure is often regarded as a *crash* and a normal closing of the peer-to-peer application as a node *failure*.

Peer-to-peer systems should be fault tolerant and be able to handle a high level of churn. It is understood that unstructured P2P systems are the most resilient to failures. Hybrid systems are vulnerable to cluster heads failures. DHT-based systems perform quite well under moderate levels of churn, however if its level increases they are not able to conduct repairs in time and the system performance quickly degrades [199]. Few DHT-based systems are specifically designed to handle a high level of churn, among them are Bamboo [200] and MSPastry [201].

2) Load balancing: Balancing of the load among the nodes in the discovery network is one of the fundamental tasks that a designer of any large decentralized system has to solve. This task is closely related to other issues like topology mismatching, bootstrapping or hot spots avoidance. Ideally the distribution of data in the network should depend on many parameters like the bandwidth available to the node, its resource capacities and time spent in the network. One would want to distribute the data in the discovery network so that nodes with largest bandwidth serve most of the search requests; nodes with large resource capabilities do the caching and the critical data is replicated on the most stable nodes. In reality it is not always easy to obtain these parameters. The common compromise is to aim at even load distribution between all nodes/cluster heads in the network.

Most of the strategies dedicated for load balancing are designed for DHT-based peer-to-peer systems. However, they can be adapted for other types of P2P networks as well. DHT-based systems provide a natural way for load balancing, by choosing random identifies for the indexed objects. However, such distribution can result in $O(\log N)$ imbalance [202]. To solve this problem a number of alternative methods have been suggested. The concept of a virtual server [202], [203], as the basic unit of load movement, allows to redistribute the load between node in DHT and eventually leads to the near-optimum load on whole network. The control over the distribution of the key address space to nodes or rearranging positions of the nodes in the overlay [204] also effects the load balancing. Load balancing in hybrid and unstructured P2P networks can also be achieved by the use of search methods designed with this purpose in mind [205], [41].

3) Flash crowds: This phenomenon occurs in a network when a certain resource catches the attention of a large number of users and consecutively gets a high amount of requests, generating a huge amount of traffic that can lead to the failure of the node hosting this resource or the performance degradation of the corresponding part of the network. Most systems try to avoid this phenomenon, for example the PROOFS system [206] introduces an additional dynamic overlay constructed from randomly selected nodes in unstructured P2P fashion. The search in the overlay is done for objects that can not be retrieved from the overwhelmed servers via randomized scoped version of flooding. However, BitTorrent (Section VII-F) embraces flash crowds as due to the system's organization users only benefit from this phenomenon as it enables very fast download of the "flashed" resource without overburdening any of the network nodes.

4) Misbehavior in P2P networks: Popular peer-to-peer systems also face issues related to the misuse or abuse of the system by some users. One of the problems that popular P2P systems face is *pollution*. It is a sabotage technique that first tampers with a valid file, so that the content become unusable and then uploads the damaged file in large quantities into P2P network. When a user wants to download a file, he has large chances to encounter over and over again the unreadable file, the user's frustration increases and in the end he might drop the search or even abandon the system. KaZaa suffers from this problem in particular. J. Liang *et al.* [207] in their work study the pollution in the FastTrack network and suggests a number of anti-pollution mechanisms. S. Marti *et al.* [208] present a voting-based reputation system that significantly reduce the destructive effect of malicious nodes in the network while introducing insignificant overhead. The authors show that their method can reduced the number of failed transactions by a factor of 20.

Another problem that is encountered by peer-to-peer networks is the large amount of *free riders*. Free riders are nodes that do not contribute to the pool of shared resources, but only download from it. There exist a range of possible solutions. For example, eMule employs a "Queue and Credit" system that regulated the time the user waits to download a file from a particular peer and based on the upload/download between these two peers [16]. Selfish Link-based InCentive (SLIC) mechanism [209] tries to solve this problem in unstructured P2P systems by encouraging nodes to share data at the exchange of better service.

XI. Conclusions

We have seen that a massive amount of work has already been carried out towards development of efficient service and resource discovery systems and frameworks. We are now in the position to conclude our review by mapping the various systems discussed above to the taxonomy given in Sections I and II. A summary of this mapping is given in Table IV. We see, as was already evident above, that the majority of the systems developed thus far have been targeted towards large-scale use in the fixed Internet. Most of these systems are based on peer-to-peer overlays with a distributed architecture. Resiliency to churn and other forms of network dynamics are key for such systems, and decentralized solutions have good characteristics in this regard. For more controlled environments, such as enterprise networks, more centralized, client-server architectures are also viable. While they tend to be more fragile, such systems do provide small response times and are quite simple to implement efficiently. Finally, we note that fairly few systems have targeted the local scale, especially in the resource constrained and wireless domains. Existing systems in this space tend to use unstructured and decentralized architectures as these can be implemented with little overhead.

However, despite the amount of work already carried out, much remains to be done. Especially solutions for large-scale wireless and hybrid networks are almost completely missing. The increased interest in wireless mesh networks will certainly make this need more acute in the near future. It is not yet clear what kind of requirements come from the ubiquitous computing scenarios. Should the visions of large-scale sensor and actuator networks be realized, none of the existing, well-established discovery systems would really be suitable due to the resource constraints and potentially complex queries.

Acknowledgment

We would like to thank DFG (Deutsche Forschungsgemeinschaft) and RWTH Aachen for the partial financial support through UMIC-excellence cluster funding. Two of us (EM and

Criterion \ Name	Bonjour	SLP	Jini	UPnP	TinyDB	JXTA	eDonkey	KaZaa	BitTorrent	Freenet	Napster	Gnutella
1. Type												
1.1. Fixed	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
1.2. Wireless	✓			✓	✓	✓						
1.3. Constraint	✓											
2. Architecture												
2.1. Client-server		✓	✓				✓				✓	
2.2. Peer-to-peer	✓			✓		✓	✓	✓	✓	✓	✓	✓
2.3. Hybrid						✓		✓				
3. Scale												
3.1. Local	✓			✓	✓							
3.2. Enterprise		✓	✓									
3.3. Internet						✓	✓	✓	✓	✓	✓	✓
4. Search												
4.1. Unstructured												
4.1.1. Uninformed	✓			✓			✓	✓				
4.1.2. Informed									✓	✓		✓
4.2. Structured		✓	✓		✓		✓				✓	
4.3. Hybrid						✓						

TABLE IV
MAPPING OF THE SERVICE DISCOVERY TAXONOMY ON THE REAL SYSTEMS.

JR) also acknowledge a previous support from the graduate school scholarships.

REFERENCES

[1] W. K. Edwards, "Discovery systems in ubiquitous computing," *IEEE Pervasive Computing*, vol. 5, no. 2, pp. 70–77, April 2006.

[2] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive Computing*, vol. 04, no. 4, pp. 81–90, October 2005.

[3] K. Vanthournout, G. Deconinck, and R. Belmans, "A taxonomy for resource discovery," *Personal Ubiquitous Computing*, vol. 9, no. 2, pp. 81–89, March 2005.

[4] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, no. 2, pp. 72–93, March 2004.

[5] C. Soanes and A. Stevenson, Eds., *Oxford Dictionary of English*. Oxford University Press, August 2005.

[6] T. Klingberg and R. Manfredi, "Gnutella protocol specification." http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, [last visited on 02.07.2007].

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," RFC 2165, June 1997.

[8] A. Lipsman, "Global napster usage plummets, but new file-sharing alternatives gaining ground," Jupiter Media Metrix, July 2001, http://www.comscore.com/press/release.asp?id=249, [last visited on 02.07.2007].

[9] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, vol. 258, Cambridge, MA, USA, March 2002, p. 263.

[10] M. Stokes, "Gnutella2 specifications part one." http://www.gnutella2.com/index.php/Main_Page#Gnutella2_Developer_Network [last visited on 02.07.2007].

[11] T. Hargreaves, "The fasttrack protocol." http://cvs.berlios.de/cgi-bin/viewcvs.cgi/gift-fasttrack/giFT-FastTrack/PROTOCOL?rev=HEAD&content-type=text/vnd.viewcvs-markup (links go as /gift-fasttrack/giFT-FastTrack/PROTOCOL) [last visited on 02.07.2007].

[12] Rahm, E. and Bernstein, P., "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

[13] R. Mueller, G. Alonso, and D. Kossmann, "SwissQM: Next generation Data Processing in Sensor Networks," in *Proceedings of the 3rd Biennial Conference on Innovative Database Research (CIDR2007)*, vol. 7, Asilomar, California, USA, January 2007.

[14] S. Helal, N. Desai, and V. Verma, "Konark-a service discovery and delivery protocol for ad-hoc networks," in *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, vol. 3, 2003, nOT FINISHED.

[15] P. E. Engelstad and Y. Zheng, "Evaluation of service discovery architectures for mobile ad hoc networks," in *WONS '05: Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services (WONS'05)*, Washington, DC, USA, January 2005, pp. 2–15.

[16] H. Breitkreuz, "emule p2p system." http://www.emule-project.net/home/perl/general.cgi?l=1, [last visited on 02.07.2007].

[17] I. Clarke, "Freenet home page." http://www.freenetproject.org/, [last visited on 02.07.2007].

[18] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, A. Wollrath, B. O'Sullivan, and A. Wollrath, *Jini Specification*, 1st ed. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[19] B. Miller, T. Nixon, C. Tai, and M. Wood, "Home networking with Universal Plug and Play," *IEEE Communications Magazine*, vol. 39, no. 12, pp. 104–109, December 2001.

[20] L. A. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, March 2003.

[21] S. Networks, "Kazaa." http://www.kazaa.com/us/index.htm, [last visited on 02.07.2007].

[22] "Bonjour," 2005, Technology Brief, http://images.apple.com/macosx/pdf/MacOSX_Bonjour_TB.pdf [last visited 23.05.2007].

[23] D. Salomon, *Data compression: the complete reference*. New York, NY, USA: Springer-Verlag New York, Inc., 1998.

[24] M. Cokus and D. Winkowski, "XML Sizing and Compression Study For Military Wireless Data," *XML Conference and Exposition*, pp. 8–13, December 2002.

[25] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[26] S. Rhea and J. Kubiatowicz, "Probabilistic Location and Routing," in *Proceedings of IEEE INFOCOM*, vol. 3, July 2002, pp. 1248–1257, New York, USA.

[27] P. Mockapetris and K. J. Dunlap, "Development of the domain name system," in *SIGCOMM'88: Symposium proceedings on Communications architectures and protocols*, Stanford, California, United States, August 1988.

[28] T. Howes and M. Smith, *LDAP: programming directory-enabled applications with lightweight directory access protocol*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 1997.

[29] U. Black, *ISDN and SS7: architectures for digital signaling networks*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1997.

[30] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," *Request for Comments*, vol. 2543, March 1999.

[31] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *Proceedings of the SOSP'01*, Chateau Lake Louise, Banff, Canada, October 2001.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proceedings of the ACM SIGCOMM*, San Diego, CA, USA, August 2001, pp. 161–172.

[33] S. Deering, "Host Extension for IP Multicasting," *IETF Request for Comment RFC1112, Aug*, 1989.

[34] A. El-Sayed, V. Roca, L. Mathy, R. Center, and F. INRIA, "A survey of proposals for an alternative group communication service," *Network, IEEE*, vol. 17, no. 1, pp. 46–51, January 2003.

[35] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *Proceedings of the 11th International Conference on Information and Knowledge Management*, McLean, VA, USA, November 2002, pp. 300–307.

[36] A. Ganesh, A. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 139–149, February 2003.

[37] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[38] C. Partridge, T. Mendez, and W. Milliken, "Host Anycasting Service," *Request For Comments*, vol. 1546, November 1993.

[39] T. Hardie, "Distributing Authoritative Name Servers via Shared Unicast Addresses," RFC 3258, April 2002.

[40] A. O'Connor, C. Brady, P. Byrne, and A. Olivré, "Characterising the eDonkey Peer-to-Peer File Sharing Network," Computer Science Department, Trinity College Dublin, Ireland, Tech. Rep., 2004.

[41] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS'02: Proceedings of the 16th international conference on Supercomputing*, New York, USA, June 2002.

[42] E. Meshkova, J. Riihijärvi, and P. Mähönen, "Evaluation of Dynamic Query Abolishment Methods in Heterogeneous Networks," in *Proceedings of the IEEE Globecom '06*, November 2006.

[43] S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, 2nd ed. Upper Saddle river, New Jersey 07458, USA: Pearson Education, 2003, pp. 73–81.

[44] ——, *Artificial Intelligence A Modern Approach*, 2nd ed. Upper Saddle river, New Jersey 07458, USA: Pearson Education, 2003, pp. 95–136.

[45] D. Tsoumakos and N. Roussopoulos, "A Comparison of Peer-to-Peer Search Methods," in *Proceedings of the WebDB'03*, June 2003, pp. 61–66, san Diego, USA.

[46] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," in *Proceedings of the 16th ACM International Conference on Supercomputing (ICS)*, June 2002, pp. 84–95, New York, USA.

[47] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Transactions on Computer Systems (TOCS)*, vol. 17, no. 2, pp. 41–88, 1999.

[48] S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, 2nd ed. Upper Saddle river, New Jersey 07458, USA: Pearson Education, 2003, pp. 165–171.

[49] S. Golomb and L. Baumert, "Backtrack programming," *Journal of the ACM (JACM)*, vol. 12, no. 4, pp. 516–524, October 1965.

[50] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, p. 671, 1983.

[51] C. C. H. Wang, T. Lin and Y. Shen, "Dynamic Search in Peer-to-Peer Networks," in *Proceedings of the 13th international World Wide Web Conference*, May 2003, pp. 332–333, New York, USA.

[52] M. Dorigo, *Ant Colony Optimization*. MIT Press, 2004.

[53] J. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, Mass. : MIT Press, December 1992.

[54] A. D. N. Ganguly, G. Canright, "Design Of An Efficient Search Algorithm For P2P Networks Using Concepts From Natural Immune Systems," in *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*, September 2004, pp. 491–500, birmingham, UK.

[55] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed., M. Horton, Ed. Prentice Hall International, Inc., 1999.

[56] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, July 2002, pp. 5–14.

[57] L. X. Z. Zhuang, Y. Liu and L. Ni, "Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks," in *Proceedings of the International Conference on Parallel Processing (ICPP 03)*, April 2003, pp. 171–178, Kaohsiung, Taiwan.

[58] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, "Search in power-law networks," *Physical Review*, vol. 64, no. 4, pp. 46 135–46 143, Sep 2001.

[59] D. Menascé and L. Kanchanapalli, "Probabilistic scalable p2p resource location services," in *Proceedings of the ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 2, Marina del Rey, CA, USA, September 2002, pp. 48–58.

[60] N. R. D. Tsoumakos, "Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks," in *Proceedings of the International Conference on Parallel Processing (ICPP 03)*, Linkoping, Sweden, September 2003.

[61] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of the The 22nd International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, July 2002.

[62] M. W. G. Sakaryan and H. Unger, "Search Methods in P2P Networks: a Survey," in *Proceedings of the Innovative Internet Community Systems (I2CS 2004)*, June 2003, Guadalajara, Mexico.

[63] C. Rohrs, "Query routing for Gnutella network," May 2002, http://www.limewire.com/ developer/query_routing/keyword%20routing.htm [last time visited on 15.02.2008].

[64] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," in *Proceedings of the 22nd International Conference on Distributed Systems*, Vienna, Austria, July 2002, p. 5.

[65] R. Morselli, B. Bhattacharjee, M. Marsh, and A. Srinivaran, "Efficient Lookup on Unstructured Topologies," *IEEE Journal on selected areas in communications*, vol. 25, no. 1, pp. 62–72, January 2007.

[66] M. Iles and D. Deugo, "A Search for Routing Strategies in a Peer-to-Peer Network Using Genetic Programming," in *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Suita, Japan, October 2002, p. 341.

[67] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, and J. Vuori, "Resource Discovery in P2P Networks Using Evolutionary Neural Networks," in *Proceedings of the AISTA*, November 2004, Luxembourg.

[68] Z. Zhuang, Y. Liu, and L. Xiao, "Dynamic layer management in super-peer architectures," in *Proceedings of the 2004 international Conference on Parallel Processing (ICPP 04)*, Montreal, Quebec, Canada, August 2004.

[69] B. Loo, R. Huebsch, I. Stoica, and J. Hellerstein, "The case for a hybrid P2P search infrastructure," in *The proceedings of IPTPS*, La Jolla, CA, USA, February 2004.

[70] G. Sakaryan, M. Wulff, and H. Unger, "Search methods in P2P networks: a survey," in *Proceedings of I2CS-Innovative Internet Community Systems (I2CS 2004)*, Guadalajara, Mexico, June 2004.

[71] X. Liu, J. Wang, and S. Vuong, "A category overlay infrastructure for peer-to-peer content search," in *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, Colorado, USA, April 2005.

[72] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, m. Palmer, and T. Risch, "Edutella: a p2p networking infrastructure based on rdf," in *WWW'02: Proceedings of the 11th international conference on World Wide Web*, Honolulu, Hawaii, USA, May 2002.

[73] H. Unger and M. Wulff, "Cluster-building in p2p-community networks," in *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, Cambridge, USA, November 2002, pp. 685–690.

[74] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, San Francisco, USA, March 2003.

[75] E. Cohen, A. Fiat, and H. Kaplan, "Associative search in peer-to-peer networks: Harnessing latent semantics," in *Proceedings of IEEE INFOCOM*, vol. 2, San Francisco, March 2003, pp. 1261–1271.

[76] A. Crespo and H. Garcia-Molina, "Semantic Overlay Networks for P2P Systems," *Lecture Notes in Computer Science*, vol. 3601, pp. 1–13, 2005.

[77] W. Müller and A. Henrich, "Fast retrieval of high-dimensional feature vectors in p2p networks using compact peer data summaries," in *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, Berkeley, California, USA, November 2003.

[78] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-peer information retrieval using self-organizing semantic overlay networks," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. Karlsruhe, Germany, August 2003.

[79] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, September 1990.

[80] K. Aberer, P. Cudre-Mauroux, M. Hauswirth, and T. Van Pelt, "GridVine: Building Internet-Scale Semantic Overlay Networks," in *International Semantic Web Conference*, Hiroshima, Japan, November 2004.

[81] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid: a self-organizing structured p2p system," *SIGMOD Rec.*, vol. 32, no. 3, pp. 29–33, September 2003.

[82] A. Löser, S. Steffen, and C. Tempich, "Semantic Social Overlay Networks," *IEEE Journal on selected areas in communications*, vol. 25, no. 1, pp. 5–14, January 2007.

[83] H. Balakrishnan, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, February 2003.

[84] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica, "Complex Queries in DHT-based Peer-to-Peer Networks," in *1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, Cambridge, MA, USA, March 2002.

[85] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," in *Proceedings of International Middleware Conference*, Rio de Janeiro, Brazil, June 2003, pp. 21–40.

[86] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica, "Querying the Internet with PIER," in *Proceedings of 19th International Conference on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.

[87] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *Proceedings of SIGCOMM 2003*, Karlsruhe, Germany, August 2003, pp. 381–394.

[88] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 SIGCOMM conference*, San Diego, CA, USA, August 2001, pp. 149–160.

[89] 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 *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, El Paso, Texas, United States, May 1997, pp. 654–663.

[90] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, California, USA, March 2004.

[91] P. Flocchini, A. Nayak, and M. Xie, "Enhancing Peer-to-Peer Systems Through Redundancy," *IEEE Journal on selected areas in communications*, vol. 25, no. 1, pp. 15–24, January 2007.

[92] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS using Chord," in *In the Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, USA, March 2002.

[93] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, Banff, Alberta, Canada, October 2001, pp. 202–215.

[94] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. 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.

[95] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *SPAA '97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, Newport, Rhode Island, USA, June 1997, pp. 311–320.

[96] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 1–14, December 2002.

[97] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," in *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, Cambridge, Massachusetts, United States, November 2000, pp. 190–201.

[98] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, USA, January 2002.

[99] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in *PODC'02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, Monterey, California, July 2002, pp. 183–192.

[100] H. Siegel, "Interconnection Networks for SIMD Machines," *Computer*, vol. 12, no. 6, pp. 57–65, June 1979.

[101] M. Kaashoek and D. Karger, "Koorde: A simple degree-optimal distributed hash table," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, USA, February 2003.

[102] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[103] N. Inc, "Napster home page," http://www.napster.com [last time visited on 15.02.2008].

[104] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, August 2003, pp. 407–418.

[105] S. Osokine, "The Flow Control Algorithm for the Distributed "Broadcast-Route" Networks with Reliable Transport Links," 2001, http://www.grouter.net/gnutella/flowcntl.htm [last visited on 02.07.2007].

[106] S. company, "Morpheus p2p client." http://www.morpheus.com/, [last visited on 02.07.2007].

[107] T. M. R. Hoffman, "Slyck – file sharing news and info." http://www.slyck.com/, [last visited on 02.07.2007].

[108] J. Liang, R. Kumar, and K. Ross, "The kazaa overlay: A measurement study," in *Proceedings of the 19th IEEE Annual Computer Communications Workshop*, Bonita Springs, Florida, USA, October 2004.

[109] A. Klimkin, "*unofficial* edonkey protocol specification v0.6.2." http://heanet.dl.sourceforge.net/sourceforge/pdonkey/eDonkey-protocol-0.6.2.html, [last visited on 02.07.2007].

[110] T. mlDonkey developing team, "mldonkey home page." http://mldonkey.sourceforge.net/MLdonkey, [last visited on 02.07.2007].

[111] I. Clarke, S. Miller, T. Hong, O. Sandberg, and B. Wiley, "Protecting free expression online with Freenet," *IEEE Internet Computing*, vol. 6, no. 1, pp. 40–49, January 2002.

[112] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC 2000)*, Portland, OR, United States, May 2000, pp. 163 – 170.

[113] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, vol. 6, Berkeley, CA, USA, June 2003.

[114] ——, "Bittorrent p2p system." http://www.bittorrent.com/index.html, [last visited on 02.07.2007].

[115] P. Soto, "Mp2p – peer-to-peer protocol." http://www.mp2p.net/mp2p.html, [last visited on 02.07.2007].

[116] N. Modus, "Dc++ – open source client for the direct connect network," http://dcplusplus.sourceforge.net/, [last visited on 02.07.2007].

[117] T. Mennecke, "Slyck news - winMX the beginning, the middle, the end," http://www.slyck.com/news.php?story= 940, [last visited on 02.07.2007].

[118] N. Modus, "Opennap: Open source napster server," http://opennap.sourceforge.net/, [last visited on 02.07.2007].

[119] A. Treves, "Ares." http://aresgalaxy.sourceforge.net/, [last visited on 02.07.2007].

[120] A. P. development team, "Anatomic p2p system." http://anatomic.berlios.de/index2.php, [last visited on 02.07.2007].

[121] C. Ding, S. Nutanong, and R. Buyya, "Peer-to-Peer Networks for Content Sharing," GRIDS-TR-2003-7, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Tech. Rep., December 2003.

[122] T. giFT project team, "The gift project," http://gift.sourceforge.net/, [last visited on 02.07.2007].

[123] M. Stokes., "Shareaza p2p client." http://www.shareaza.com/, [last visited on 02.07.2007].

[124] J. Konishi, N. Wakamiya, and M. Murata, "Proposal and Evaluation of a Cooperative Mechanism for Pure P2P File Sharing Networks," in *Proceedings of the 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT2006)*, Osaka, Japan, January 2006, pp. 33–47.

[125] S. Lui and S. Kwok, "Interoperability of peer-to-peer file sharing protocols," *ACM SIGecom Exchanges*, vol. 3, no. 3, pp. 25–33, June 2002.

[126] J. Waldo, "The Jini architecture for network-centric computing," *Communications of the ACM*, vol. 42, no. 7, pp. 76–82, July 1999.

[127] P. Bhagwat, "Bluetooth: technology for short-range wireless applications," *IEEE Internet Computing*, vol. 5, no. 3, pp. 96–103, May 2001.

[128] S. Bluetooth, "Specification of the Bluetooth System-Core," *Technical Specification Version*, vol. 2.1, July 2007, http://www.bluetooth.com/bluetooth/.
[129] "UPnP Forum," http://www.upnp.org/ [last time visited on 15.01.2007].
[130] "UPnP device architecture version 1.0," July 2006, document version 1.0.1, http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf [last visited 07.07.2007.
[131] S. Cheshire, B. Aboba, and G. E., "Dynamic configuration of ipv4 link-local addresses," RFC 3927, May 2005.
[132] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, "Simple Service Discovery Protocol/1.0, Operating without an Arbiter," Internet-Draft (expired), http://www.cs.colorado.edu/ rhan/CSCI_7143_002_Fall_2001/Papers/draft_cai_ssdp_v1_03.txt [last visited on 02.07.2007], October 1999.
[133] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen, "Simple Object Access Protocol (SOAP) 1.2," W3C Recommendation, June 2003.
[134] J. Cohen, S. Aggarwal, and Y. Goland, "General Event Notification Architecture Base: Client to Arbiter," Internet-Draft (expired), http://tools.ietf.org/html/draft-cohen-gena-client-00 [last visited on 02.07.2007], June 1999.
[135] K. Mills and C. Dabrowski, "Adaptive jitter control for UPnP M-search," in *Proceedings of the IEEE International Conference on Communications (ICC'03)*, Anchorage, AK, USA, May 2003.
[136] J. M. Sanchez Santana, M. Petrova, and P. Mähönen, "UPnP service discovery for heterogeneous networks," in *Proc. of IEEE PIMRC*, September 2006, Helsinki.
[137] J. Newmarch, "A RESTful approach: clean UPnP without SOAP," *Proceedings of the 2nd IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 134–138, January 2005.
[138] R. Droms *et al.*, "Dynamic Host Configuration Protocol," RFC 2131, March 1997.
[139] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," IETF Draft, http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt [last visited on 02.07.2007], August 2006.
[140] ——, "Multicast DNS," IETF Draft, http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt [last visited on 02.07.2007], August 2006.
[141] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J. Hugly, E. Pouyoul, and B. Yeager, "Project JXTA 2.0 Super-Peer Virtual Network," Sun Microsystem White Paper. Available at www.jxta.org/project/www/docs, 2003.
[142] C. Goldfarb and P. Prescod, *The XML handbook*, 4th ed. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001.
[143] A. Arora, C. Haywood, and K. Pabla, "JXTA for J2ME–Extending the Reach of Wireless with JXTA Technology," March 2002.
[144] H. Fang, L. Ko, and H. Fang, "A Design of Service-based P2P Mobile Sensor Networks," in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC)*, Taichung, Taiwan, June 2006.
[145] G. Antoniu, M. Jan, and D. Noblet, "Enabling JXTA for High Performance Grid Computing," INRIA, IRISA, Rennes, France, Research Report RR-5488, Tech. Rep., February 2005.
[146] U. Kozat and L. Tassiulas, "Network layer support for service discovery in mobile ad hoc networks," in *Proceedings of INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 3, San Francisco, CA, USA, March 2003, pp. 1965–1975.
[147] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual ring routing: network routing inspired by dhts," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 351–362, 2006.
[148] L. Cheng, "Service advertisement and discovery in mobile ad hoc networks," in *Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments, in conjunction with the ACM 2002 Conference on Computer Supported Cooperative Work*, New Orleans, Louisiana, USA, November 2002, pp. 16–20.
[149] C. Ververidis and G. Polyzos, "Routing Layer Support for Service Discovery in Mobile Ad Hoc Networks," in *Proceedings of the 3rd Intl Conf. on Pervasive Computing and Communications Workshops*, Kauai Island, Hawaii, USA, March 2005, pp. 258–262.
[150] O. Ratsimor, D. Chakraborty, A. Joshi, and T. Finin, "Allia: alliance-based service discovery for ad-hoc environments," in *WMC'02: Proceedings of the 2nd international workshop on Mobile commerce*, Atlanta, Georgia, USA, September 2002.
[151] C. Huang, T. Hsu, and M. Hsu, "Network-Aware P2P File Sharing over the Wireless Mobile Networks," *IEEE Journal on selected areas in communications*, vol. 25, no. 1, pp. 5–14, January 2007.
[152] M. Nidd, "Service discovery in DEAPspace," *IEEE Personal Communications*, vol. 8, no. 4, pp. 39–45, August 2001.
[153] F. Sailhan and V. Issarny, "Scalable service discovery for manet," in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, Kauai Island, Hawaii, USA, March 2005, pp. 235–244.
[154] Z. Shelby, P. Mähönen, J. Riihijärvi, O. Raivio, and P. Huuskonen, "NanoIP: the zen of embedded networking," in *Proceedings of ICC 2003*, Seattle, WA, USA, May 2003.
[155] B. Martin and B. Jano, "WAP binary XML content format. W3C note," *World Wide Web Consortium (June). Cambridge, MA*, 1999.
[156] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.
[157] A. Woo, S. Madden, and R. Govindan, "Networking support for query processing in sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 47–52, 2004.
[158] S. Shakkottai, T. Rappaport, and P. Karlsson, "Cross-layer design for wireless networks," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 74–80, October 2003.
[159] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *6th Annual International Conference on Mobile Computing and Networking (MOBICOM 2000)*, Boston, MA, USA, August 2000, pp. 56–67. [Online]. Available: citeseer.ist.psu.edu/intanagonwiwat00directed.html
[160] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, Banff, Alberta, Canada, October 2001, pp. 146–159.
[161] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, "TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks," in *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, Boston, MA, USA, December 2002.
[162] I. Solis and K. Obraczka, "In-network aggregation trade-offs for data collection in wireless sensor networks," *International Journal of Sensor Networks*, vol. 1, no. 3, pp. 200–212, 2006.
[163] P. Schramm, E. Naroska, P. Resch, J. Platte, H. Linde, G. Stromberg, and T. Sturm, "A service gateway for networked sensor systems," *IEEE Pervasive computing*, vol. 3, no. 1, pp. 66–74, 2004.
[164] T. Luckenbach, P. Gober, S. Arbanowski, A. Kotsopoulos, and K. Kim, "TinyREST: A Protocol for Integrating Sensor Networks into the Internet," in *Proceedings of REALWSN 2005*, Stockholm, Sweden, June 2005.
[165] D. Valetchev and I. Frankow, "Service Gateway Architecture for a Smart Home," *IEEE Communications Magazine*, vol. 40, no. 4, pp. 126–132, April 2002.
[166] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.
[167] H. Gupta, Z. Zhou, S. Das, and Q. Gu, "Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution," *IEEE/ACM Transactions on Networking*, vol. 14, no. 1, pp. 55–67, 2006.
[168] W3C, "Web Services Activity," http://www.w3.org/2002/ws/ [last visited 20.06.07].
[169] P. Erdös and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.
[170] D. Watts and S. Strogatz, "Collective dynamics of'small-world'networks," *Nature*, vol. 393, no. 6684, pp. 409–10, June 1998.
[171] A. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, October 1999.
[172] J. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, p. 845, 2000.
[173] ——, "The Small-World Phenomenon and Decentralized Search," *SIAM News*, vol. 37, no. 3, pp. 1–2, April 2004.
[174] ——, "Complex Networks and Decentralized Search Algorithms," in *Proceedings of the International Congress of Mathematicians (ICM)*, Madrid, Spain, August 2006.
[175] H. Thadakamalla, R. Albert, and S. Kumara, "Search in weighted complex networks," *Physical Review E*, vol. 72, no. 6, p. 66128, December 2005.
[176] M. Penrose, *Random Geometric Graphs*. Oxford University Press, 2003.

[177] H. Thadakamalla, R. Albert, and S. Kumara, "Search in Spatial Scale-free Networks," *New Journal of Physics*, vol. 9, no. 6, June 2007.

[178] P. Mähönen, M. Petrova, and J. Riihijärvi, "Applications of topology information for cognitive radios and networks," in *Proceedings of IEEE DySPAN 2007*, April 2007.

[179] J. Riihijärvi, P. Mähönen, and M. Rübsamen, "Characterizing wireless networks by spatial correlations," *Communications Letters, IEEE*, vol. 11, no. 1, pp. 37–39, January 2007.

[180] Z. Xu, C. Tang, and Z. Zhang, "Building topology-aware overlays using global soft-state," in *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03)*, Providence, Rhode Island, USA, May 2003, pp. 500–508.

[181] B. Bollobás, *Random Graphs*, 2nd ed. Cambridge University Press, 2001.

[182] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA, October 2003, pp. 314–329.

[183] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, June 2005.

[184] M. Jelasity, A. Kermarrec, and M. van Steen, "The peer sampling service: experimental evaluation of unstructured gossip-based implementations," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Toronto, Ontario, Canada, October 2004, pp. 79–98.

[185] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search," in *Proceedings of the 2007 ACM SIGCOMM Conference*, Kyoto, Japan, August 2007.

[186] L. Ni and Y. Liu, "Efficient Peer-to-Peer Overlay Construction," in *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, Beijing, China, September 2004, pp. 314–317.

[187] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, October 2002.

[188] Y. Liu, Z. Zhuang, L. Xiao, and L. Ni, "A distributed approach to solving overlay mismatching problem," in *Proceedings on the 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004, pp. 132–139.

[189] Y. Liu, X. Liu, L. Xiao, L. Ni, and X. Zhang, "Location-aware topology matching in p2p systems," in *Proceedings of IEEE INFOCOM'04*, Hong Kong, China, March 2004, pp. 2220–2230.

[190] Y. Liu, L. Xiao, and L. Ni, "Building a scalable bipartite P2P overlay network," in *Proceedings on the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, New Mexico, April 2004.

[191] D. Mills, "RFC 1305: Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, March 1992.

[192] V. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," in *Proceedings of ACM SIGCOMM'01*, vol. 31, no. 4, San Diego, CA, USA, August 2001, pp. 173–185.

[193] P. F. Tsuchiya, "The landmark hierarchy: a new hierarchy for routing in very large networks," in *SIGCOMM'88: Symposium proceedings on Communications architectures and protocols*, Stanford, California, United States, August 1988, pp. 35–42.

[194] A. Gupta, D. Agrawal, and A. El Abbadi, "Approximate range selection queries in peer-to-peer systems," in *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, USA, January 2003.

[195] F. Cuenca-Acuna, C. Peery, R. Martin, and T. Nguyen, "PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities," in *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, Washington, USA, June 2003, pp. 236–246.

[196] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris, "On the Feasibility of Peer-to-Peer Web Indexing and Search," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, USA, February 2003, pp. 207–215.

[197] I. Witten, A. Moffat, and T. Bell, "Managing Gigabytes: Compressing and Indexing Documents and Images," *IEEE Transactions on Information Theory*, vol. 41, no. 6, November 1995.

[198] E. Demaine, A. López-Ortiz, and J. Munro, "Adaptive set intersections, unions, and differences," in *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, San Francisco, CA, United States, January 2000, pp. 743–752.

[199] J. Li, J. Stribling, T. Gil, R. Morris, and F. Kaashoek, "Comparing the performance of distributed hash tables under churn," in *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, La Jolla, CA, USA, February 2004.

[200] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, USA, June 2004.

[201] M. Castro, M. Costa, and A. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, Florence, Italy, June 2004, p. 9.

[202] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, USA, February 2003.

[203] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *Proceedings of INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Hong Kong, China, March 2004.

[204] D. R. Karger and M. Ruhl, "Simple efficient load balancing algorithms for peer-to-peer systems," in *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, Barcelona, Spain, June 2004, pp. 36–43.

[205] S. Waterhouse, "JXTA Search: Distributed Search for Distributed Networks," Sun Microsystems Whitepaper http://search.jxta.org/JXTAsearch.pdf [last visited on 06.11.2007], 2001.

[206] A. Stavrou, D. Rubenstein, and S. Sahu, "A lightweight, robust P2P system to handle flash crowds," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 6–17, January 2004.

[207] J. Liang, R. Kumar, Y. Xi, and K. Ross, "Pollution in p2p file sharing systems," in *IEEE INFOCOM'05*, Miami, Florida, USA, March 2005.

[208] S. Marti and H. Garcia-Molina, "Limited reputation sharing in P2P systems," in *Proceedings of the 5th ACM conference on Electronic commerce*, New York, USA, May 2004, pp. 91–101.

[209] Q. Sun and H. Garcia-Molina, "SLIC: a selfish link-based incentive mechanism for unstructured peer-to-peer networks," in *Proceedings of the 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004, pp. 506–515.