Tailoring Service Discovery to Embedded P2P Systems*

Antonio Brogi
Dept. of Computer Science
University of Pisa, Italy
brogi@di.unipi.it

Sara Corfini
Dept. of Computer Science
University of Pisa, Italy
corfini@di.unipi.it

Thaizel Fuentes
Dept. of Computer Science
University of Pisa, Italy
fuentes@di.unipi.it

ABSTRACT

We present a service discovery architecture for embedded peer-to-peer systems, tailored to deal with low-capacity, mobile devices. The proposed discovery mechanism is built on top of the (Chord) Distributed Hash Table technology, extending it by fruitfully exploiting high-capacity devices (when available) to run non-trivial matching algorithms. The design of the service discovery architecture has been motivated by and carried over within the European Project SMEPP (Secure Middleware for Embedded Peer to Peer systems).

1. INTRODUCTION

Embedded peer-to-peer (EP2P) systems represent a new challenge in the development of software for distributed systems. EP2P systems present a high degree of *heterogeneity* (i.e., different devices with different computing power) and *autonomy* (i.e., devices enter and exit the system in an independent way), which raise important technological challenges such as decentralisation, transient communications, and a constantly changing topology. One of the objectives of the SMEPP European project is to provide a *service-oriented* model to program the interaction among devices, thus hiding low-level details that concern the supporting infrastructure. *Service discovery* is hence one of key issues in SMEPP.

We consider *heterogeneous* P2P networks where the number of low-capacity devices (e.g., PDAs, smart phones) considerably exceeds the number of high-capacity devices (e.g., PCs, laptops, servers). Devices offer services by *publishing* service *contracts* to inform other devices about the provided services. We consider *enhanced* contracts, which beside *type of service*, *signature* and *grounding*, possibly include *ontology* and/or *behaviour* information. Devices locate services either via *keyword-based* queries or via *enhanced* queries (e.g., queries requesting a specific service behaviour). Devices can join and (possibly unexpectedly) leave the network.

In this paper we propose a service discovery architecture which adapts to varying network configurations – from generic heterogeneous networks to networks consisting of low-capacity devices only – thus handling continuous connections and (possibly unexpected) disconnections of devices.

In a network consisting of low-capacity devices only, devices own limited computational resources, yet sufficient to run lightweight, trivial matching algorithms (e.g., syntactic matching of service operations and their parameters). The service discovery workload is distributed among *all* available low-capacity devices.

In heterogeneous P2P networks, there are devices with powerful computational resources available. The discovery architecture then suitably exploits *high-capacity devices* to:

- *back-up service contracts* – high-capability device store copies of service contracts published by low-capacity devices in their vicinity,
- *preserve low-capacity devices' resources* – high-capacity devices carry most of the burden of service discovery, thus saving the limited resources of low-capability devices,
- *run non-trivial matching algorithms* – due to the availability of powerful resources, high-capacity devices are in charge of executing behaviour and/or ontology aware matching algorithms.

A two-layers DHT is the key ingredient of the service discovery architecture. It consists of two overlapped Chord rings, the lower one including both low-capacity and high-capacity devices, and the upper one including high-capacity devices only. While the cost of maintaining the upper Chord ring is very limited (since it shares the keyspace and the node identifiers of the lower ring), the two-layers DHT architecture suitably distributes the discovery workload among *all* the available devices, while greatly exploiting, when available, high-capacity devices.

The rest of the paper is organised as follows. Section 2 briefly describes motivations and requirements of the SMEPP project. Section 3 presents the proposed service discovery architecture for embedded peer-to-peer systems. Related work is discussed in Section 4, while some concluding remarks are drawn in Section 5.

*Research partially supported by EU FP6-IST STREP 0333563 SMEPP.

MobiQuitous 2008, July 21-25, 2008, Dublin, Ireland.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904

2. A BRIEF INTRODUCTION TO SMEPP

SMEPP (Secure Middleware for Embedded Peer to Peer systems, `http://www.smepp.org`) is an European research project whose main objective is to develop a *service-oriented* middleware for embedded P2P systems. The SMEPP middleware will have to be secure, generic and *highly customisable*, allowing for its adaptation to different devices and domains.

We list below those SMEPP requirements which have mainly influenced the design of the service discovery architecture presented in the next Section.

(1) *Devices* – Two types of devices[1] are identified. *Low-capacity* devices (e.g., PDAs, smart phones) are devices with limited CPU/RAM performance (e.g., 200MHz/ 64MB), short-life battery and (possibly) short-range connectivity (e.g., bluetooth). *High-capacity* devices (e.g., PCs, laptops) are devices with powerful CPU/ RAM resources (e.g., 1GHz/512MB), long-life battery, and (possibly) wide-range connectivity (e.g., Wi-Fi). Yet, such a categorization is inherently dynamic, indeed a last-generation PDA (e.g., with 400MHz CPU, 128MB RAM, Wi-Fi, running on main-power) may act as high-capacity device. Each device entering the (SMEPP) network can hence be classified as either a *low-* or *high-*capacity device w.r.t. its actual resources.

(2) SMEPP API – SMEPP provides a set of primitives for dynamically joining and leaving the (SMEPP) network, for raising and receiving events, and for *publishing*, *discovering* and invoking services. The SMEPP API enables "*on-the-fly*" conversations with SMEPP services.

(3) *Service contracts* – Devices offer services and *publish* service *contracts* to inform the other devices about the provided services. A SMEPP contract consists of *service signature* (i.e., a WSDL2.0-based description of service operations and types) optionally annotated with *ontology* information, *non-functional* properties (i.e., type of service and QoS attributes), and of service *behaviour* (expressed in SMoL, a BPEL-inspired service model language).

(4) *Contract templates* – Clients submit *partially* specified contracts – the so-called *contract templates* – defining the main features of the services to be discovered.*Syntactic queries* specify service signature and/or non functional service properties. *Enhanced queries* specify ontology-annotated service signature and/or service behaviour.

(5) *Group security* – Security is bound to groups, and service discovery is carried out at the group level. Devices offer services inside groups, and a device must belong to a group in order to use the services provided by the group.

3. A P2P DISCOVERY ARCHITECTURE

Distributed hash table (DHT) technology is a key ingredient to implementing decentralised service discovery mechanisms. For instance, distributing service contracts (e.g., the *type of service* may be employed to hash them) among the nodes of a Chord DHT ring [15], each node corresponding to a device in the P2P network, is sufficient to build a basic service discovery mechanisms. Several drawbacks are yet identified:

- *no load-balancing* – the basic DHT discovery mechanisms does not distinguish between low-capacity and high-capacity devices. Indeed, low-capacity devices are always engaged in the discovery task, even when there are high-capacity devices available.
- *no best match guaranteed* – (enhanced) matching of service contracts is to be run on the device which stores contracts selected by the keyword-based DHT discovery mechanisms. If such a device is a low-capacity device, no enhanced matching can be executed, so that enhanced queries can be only partially satisfied.
- *no device disconnection management* – when a device unexpectedly disconnects from the network (e.g., due to exhausted battery), the service contracts it stores are lost, while the contracts (published by other devices) of the services it provides get the status of *dangling* contracts.

In the following subsections, we present a service discovery architecture which extends the DHT technology to suitably exploit, when available, high-capacity devices to backup service contracts, to run non-trivial matching algorithms, and to save low-capacity devices' resources.

3.1 A two-layers DHT

The proposed service discovery architecture is overviewed in Figure 1. It consists of two overlapped Chord rings, the lower one containing *all* the (low-capacity and high-capacity) devices in the P2P network, and the upper one containing the high-capacity devices only. Thus, high-capacity devices belong to both Chord rings.

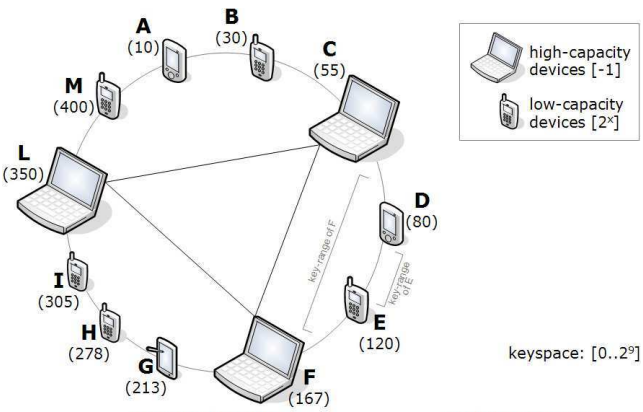

Figure 1: A two-layers DHT.

Each device is associated with a single *node identifier* for both rings, which share the keyspace. To realize the two-layers DHT, each device maintains a *routing table* (i.e., a set of links to other DHT nodes), slightly different from the the original Chord routing table, as described below:

[1] *Sensors* are not considered in this paper, as each of them is associated to a sink (a low- or high-capacity device) acting as a proxy toward the rest of the network.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904

(1) *predecessor and successor* – as in the original Chord approach, each node stores two identifiers denoting its predecessor and its successor on the lower ring. Additionally, each high-capacity device stores two further identifiers, which denote its predecessor and its successor on the upper ring (e.g., L^2 and F are the predecessor and the successor of C), while each low-capacity device stores the identifier (hcd OWNER) of the high-capacity device whose key range overlaps its key range (e.g., L is the hcd OWNER of H, since the key range of L (i.e., [168,350]) overlaps with the key range of H (i.e., [214,278]).

(2) *fingers* – as in the original Chord approach, each node stores the identifiers of other nodes (i.e, the *fingers*) in the ring to guarantee a logarithmic data lookup. Yet, we mainly devote the space available in a finger table to store the identifiers of high-capacity devices, fulfilling the remaining space with low-capacity devices' identifiers. Note that no additional space is hence needed to realize the finger tables of two Chord rings. For example, in Figure 1 the finger table of C contains high-capacity devices' identifiers only (F, L) as they suffice to entirely cover the keyspace of the Chord rings.

Given the availability of two types of devices, two Chord rings are necessary (and suffice) to properly manage the different capabilities of low-capacity and high-capacity devices. Intuitively, the lower Chord ring guarantees an efficient service discovery mechanism (w.r.t. response time and number of exchanged messages) even when there are no high-capacity devices available, while the upper Chord ring enables the suitable exploitation of high-capacity devices, if available, in the service discovery task.

Subsections 3.2 and 3.3 describe the discovery architecture when *publishing* and *searching* service contracts.

3.2 Publishing service contracts

To make the provided services accessible to the other devices in the P2P network, devices *publish* service contracts. A device providing a service first *hashes* the service contract to generate a DHT *key*, then sends (a copy of) the contract to the device responsible for that key (viz., the *contract host*), which actually publishes the service contract. To hash service contracts we use a set of *basic attributes*, which include *type of service* and *service properties* (e.g., the *group* of the service provider and *QoS*). Once a set of basic attributes is selected, *all* the contracts *must* describe each of them. The Hilbert SFC (Space Filling Curve) function [5] is employed to hash multiple contract parameters into single (Chord) DHT keys.

The device responsible for the key of the service contract to be published stores the service contract into its local (contract) repository. If such a device is a low-capacity device, then (a copy of) the contract is also stored by a high-capacity device, if available. To this end, the low-capacity device analyses its routing table to determine the closest high-capacity device in its vicinity responsible for the key and sends (a copy of) the contract to it.

[2]For the sake of readability, we use capital letters (L) to denote nodes on the Chord rings in Figure 1, thus not using their numeric identifiers (350).

A (contract) repository is a set of tuples ⟨key, contract, TTL, ProviderID⟩, where contract is the service contract, key is the DHT key generated by hashing the contract, TTL is the time to live of the contract and ProviderID is the physical address of the node providing the contract. A tuple ⟨key, contract, TTL, ProviderID⟩ is removed from the contract repository when TTL expires. Consequently, to keep *alive* a provided service, a device has to *refresh* the TTL of its contract at regular time intervals (note that RefreshTTL messages do not carry the contract). If a device receives a RefreshTTL request for an *unknown* contract (e.g., due to some changes in the network topology), then the device sends a RePublishContract request to the provider of the expiring service contract. Unexpected disconnections of devices are hence managed properly. Indeed, if a device providing a service disconnects, the devices hosting its service contract will remove it when its TTL expires, thus deleting dangling contracts. If a device hosting a service contract disconnects, another device will store the contract when the device providing the service re-publish it. Note that if the disconnected device is a low-capacity device, a copy of the service contract is also available on a high-capacity device (if any).

3.3 Discovering service contracts

To locate the services available in the P2P network, devices *query* the P2P system. We consider two types of requests:

- *syntactic queries* – i.e., queries specifying some keywords (e.g., type of service, QoS attributes, name of a service operation) which have to be included in the contracts of the selected services.
- *enhanced queries* – i.e., queries also specifying advanced features (e.g., service behaviour) which have to be satisfied by the contracts of the selected services.

We assume that *both* syntactic and enhanced queries specify (at least) the *basic attributes* (viz., type of service and service properties), that is, those service parameters used to hash service contracts (see Subsection 3.2). The service discovery architecture satisfies a query in two steps. First, the two-layers DHT locates the device hosting the service contracts which satisfy the basic query attributes. Then, a (possibly non-trivial) matching algorithm is run to select those service contracts which satisfy the complete requirements of the (syntactic/enhanced) query.

Several discovery scenaria can be distinguished with respect to the devices actually available in the P2P network and with respect to the type of the query to be satisfied.

Consider first a (low-capacity) device in a full low-capacity devices network, which submits a *syntactic* query to the P2P system. Then, the device hashes the syntactic query to generate a DHT key and sends the pair ⟨query, key⟩ to the device responsible for that key. Note that in a full low-capacity devices network, there is no upper Chord ring. If the syntactic query contains only the basic attributes of service contracts, the query is fully satisfied by the classical DHT discovery mechanisms. Indeed, the device storing the selected contracts sends them to the requesting device. If this is not the case (viz., the syntactic query contains service parameters other than the basic service attributes), the contracts need to be further matched against the complete query requirements. We assume that low-capacity devices

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904

are in charge of running lightweight, trivial matching algorithms (e.g., a syntactic matching of service operations and their parameters). Contracts are matched either by the device hosting them or by the requesting device according to their current conditions (e.g., battery status, current workload). In the former case, the contracts received by the requesting device fully match the query, in the latter case they match the basic query attributes only.

Instead, if a (low-capacity) device submits an *enhanced* query to the low-capacity P2P system, the query can be only partially satisfied. Indeed, the DHT discovery mechanisms locates the contracts satisfying the basic attributes of the query, yet, there is no (high-capacity) device capable of running the non-trivial matching algorithms necessary to fulfill the enhanced query.

Consider now a device in a *heterogeneous* P2P network, which submits a query to the P2P system. In such a scenario, the discovery task is implemented by the high-capacity devices on the upper Chord ring, thus preserving the low-capacity devices' resources. Thus, a low-capacity device, which submits a query to the P2P system, hashes the query to generate a DHT key and sends the pair ⟨query, key⟩ to the closest high-capacity device in its (Chord-)vicinity. The high-capacity device is the "access point" to the upper Chord ring, which is in charge of satisfying the query of the low-capacity device. Note that syntactic as well as enhanced queries can be fully satisfied by the high-capacity devices on the upper Chord ring, since they own the necessary resources to run non-trivial matching algorithm.

3.4 Setting up the two-layers DHT

While the proposed two-layers DHT architecture efficiently answers the devices' requests, suitably exploits (when available) the powerful resources of high-capacity devices, and adapts to varying network configurations, it entails the additional cost of maintaining the upper Chord ring. In this Subsection, we describe the operations necessary to handle connections and (possibly unexpected) disconnections[3] of devices.

We first consider a device joining the P2P network. The device analyses the routing tables of the devices in its physical vicinity to determine an overloaded fragment (w.r.t. the number of stored contracts) of the (lower) Chord ring, and gets a DHT key within such a fragment. Part of the contracts stored by the previous device on the lower Chord ring move into the new device. Additionally, if the joining device is a high-capacity device, it also retrieves (a copy of) the contracts published by the low-capacity devices overlapping its upper Chord ring fragment. For instance, consider the high-capacity node F on the two-layers Chord ring of Figure 1. Its local (contract) repository will store (a copy of) the contracts of the low-capacity devices D and E.

The most critical scenario is a huge network with low-capacity devices only, and a single high-capacity device entering the network. The arrival of such a high-capacity device would raise a huge exchange of messages (including copies of XML contracts), that would be unbearable in a low-capacity devices network. However, in order to reduce the amount of exchanged data and to avoid heavy broadcasts, we establish that each high-capacity device manages just a predefined N-hops Chord vicinity (counterclockwise). In particular, the joined node sends a GetCopyMsg message, containing N and its identifier id, to its predecessor on the lower ring. Next, the predecessor sends a copy of its contracts to the high-capacity device by using the received id, sets its high-capacity device owner to id, and then forwards, in turn, the GetCopyMsg to its respective predecessor, yet passing N-1 and id. The process continues in the same way, stopping when N becomes 0 or (in general) when another high-capacity device is found[4].

We now consider a device (properly) disconnecting from the P2P network. Then, the device distributes the contracts that it actually publishes (as node of the lower ring) among the devices on the lower Chord ring in its vicinity. Additionally, if the leaving device is a high-capacity device, it distributes the copied contracts in its local (contract) repository among the high-capacity devices on the upper Chord ring in its N-hop Chord-vicinity. Note that if there are no (other) high-capacity devices, the copied contracts are lost, the original contracts still published by other (low-capacity) devices.

Finally, we consider a device unexpectedly leaving the P2P network. On the one hand, the Chord recovery mechanism updates the ring(s) according to the new network topology, on the other hand, the TTL-based publication mechanism (previously described in Subsection 3.2) re-distributes the service contracts hosted by the disconnected device among the nodes of the updated ring(s).

3.5 A component-based architecture

This Subsection briefly presents a component-based architecture for the service discovery mechanisms described in Subsections 3.1, 3.2 and 3.3, which can be directly plugged into the SMEPP component-based architecture [14].

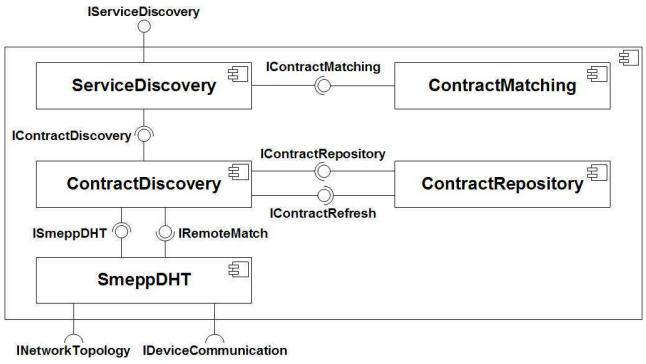

Figure 2: A component-based architecture.

As illustrated in Figure 2, the component-based architecture consists of five sub-components:

[3]We consider device *mobility* as a set of device connections and disconnections.

[4]The value of N can be suitably configured at runtime, considering also the maximum amount of contracts to be stored on the joined high-capacity device.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904

- ServiceDiscovery – it provides the interfaces PublishContract (to publish a service contract) and GetServices (to search for service contracts) to interact with the discovery component.
- ContractDiscovery – it is responsible for the generation of (Chord) DHT keys. It hashes service contracts as well as clients' queries (viz., contract templates).
- SmeppDHT – it implements the two-layers DHT previously introduced in Subsection 3.1. SmeepDHT interacts with other (existing) SMEPP components which provide information on the network topology and primitives for device communications.
- ContractRepository – it stores the service contracts published by the device. It consists of a set of tuples $\langle \mathsf{key}, \mathsf{contract}, \mathsf{TTL}, \mathsf{ProviderID} \rangle$, as previously defined in Subsection 3.2.
- ContractMatching – it provides the algorithms to match service contracts: lightweight, syntactic matching algorithms if the device is a low-capacity device, enhanced (i.e., ontology/behaviour aware) matching algorithms if the device is a high-capacity device.

The clear separation of *contract discovery* and *contract matching* functionalities in the component-based architecture in Figure 2 enables *configurability*. Indeed, for instance, different devices differently implement the ContractMatching component.

4. RELATED WORK

First, it is worth noting that well-known protocols for P2P service discovery –like UPnP[1], SLP[4], or JXTA[2]– rely on the TCP/IP stack and use broadcast/multicast advertisements. As such, they cannot hence be directly employed in multi-hop networks consisting of small devices (like SMEPP), where IP-based communication is not allowed. Moreover, in *UPnP*, *SLP* and *JXTA* service advertisements do not include behavioral or semantic descriptions of services.

We chose to build our discovery mechanism on top of Chord [15] because of Chord's simplicity and performance. Chord's data-lookup sends $O(logN)$ messages (where `N` is the number of nodes in the network), while maintaining the routing table requires (at most) $O(log^2 N)$ messages exchanged. In particular, Chord resulted more suitable to our needs than for instance *Kademlia* (which employs big routing tables and existing data is not guaranteed to be found) and *CAN* (which consume $O(d * \sqrt[d]{N})$ for data-lookup, where `d` is the number of dimension in a d-dimensional Cartesian keyspace). Moreover, the circular keyspace of Chord fits well with the Hilbert SFC[5]. Our approach to key-generation has some points in common with *Squid*[12], but we exploit high-capacity devices (when available) better than [12]. Both *Pastry* and *Tapestry* are considered to be less scalable in scenaria with frequent (dis)connections where the simpler join protocol of Chord makes a difference. A survey to the previous discussed DHT approaches is offered in [13].

Several authors employ a notion of centralized registry split into a set of cooperative high-capacity devices, each one acting as partial registry and forming so-called *backbone virtual networks*[9, 11, 16] or *super-peer networks* [6, 8, 10]. Low-capacity devices should know at least one high-capacity device in order to discover services. In absence of high-capacity devices, some low-capacity devices become overloaded or, even worse, the whole network ceases to work. In general, solutions fully based on the notion of *backbone virtual networks* scale well for networks having some high-capability devices, but they are not efficient in absence of high-capacity devices. Therefore, in general, our contribution in comparison with such approaches is to provide a fully decentralized environment, in which low-capacity devices are able to discover services and to run light matching algorithms by themselves (in absence of high-capacity)distributing the execution load among all of them (and not just among a few of them), but also fruitfully using high-capacity devices when they are available, releasing low-capacity devices roles during discovery. We now shortly discuss more in detail the main differences between [9, 6, 8, 10, 11, 16] and our approach:

Meteor-S WSDI [16] uses an ontology-based approach to organize multiple semantic registries into domains, enabling domain-based classification of Web services. Registries are distributed in a P2P network, but the *registry ontology* is managed and provided by a single peer (called gateway). The interaction with the *backbone network* is centralized by the *gateway*. If the gateway fails, the system becomes unable to work. Our proposal uses a fully decentralized approach and then is more fault-tolerant than *Meteor-S WSDI*. In *Ariadne*[11] devices in the *backbone* are able to execute non-trivial matching algorithms. Low-capacity devices discover high-capacity devices by making broadcast in the vicinity (n-hop broadcasting). When there is no high-capacity device available, one of the low-capacity starts acting as "registry" becoming considerably more overloaded than its homologous. In this sense, both *Spider*[10] and *M-Chord*[6] use approaches very similar to *Ariadne*. *PLASTIC*[9] adopts the WS-Discovery standard, extending the WSDL description with (BPEL-based) behaviour, quality of service, and context-aware information into the contract. *PLASTIC* service discovery uses a semi-distributed approach, having a backbone structured around WSD-Proxies, which act as service registries executing enhanced matching. *PLASTIC* assumes at least one WSD-Proxy in the network, while in our proposal all devices (in absence of high-capacity) are able to store contracts and can make (at least) light matching. *Spider*[10], *M-Chord*[6] and *R-Chord*[8] *super-peer networks* are developed on top of a Chord ring, and low-capacity devices act as clients of the network, sending queries and waiting for results. [8] for instance implements a semantic-based resource discovery using three overlay networks: a *Chord ring* for storing resources, a *supper-peer network* for managing resource views (viz. inferring resource schemas) and a *P2P semantic-link network*, connecting peers based on semantic relations. [8] is based on TCP/IP protocol and implies a high cost for network maintenance, therefore it is not scalable for multi-hop mobile network. While in heterogeneous networks (having considerable number of high-capacity devices) our proposal behaves similar to *Ariadne* and *Spider*, in networks formed mainly by low-capacity devices, our proposal provides a more decentralized solution, as previously discussed.

On the other hand, some authors assume a fully decentralized point of view (as we do), basing their proposal on the presence of a structured overlay network (DHT), for efficient routing by means of keyword-based queries [7, 3]. *DWSDM*

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904

[7] is a P2P architecture for Web Service Discovery based on DHT. The main difference with our proposal is that *DWSDM* is not device-capacity aware, since it assumes all devices equivalent and it does not consider in the architecture any type of non-trivial matching. *Hybrid-Chord* [3] has some point in common with our proposal, it is based on the idea on several overloaded Chord rings, with the goal of forcing data replication and in that way retrieving data in less hops in comparison with a single Chord-ring. Differently from us, *Hybrid-Chord* assigns several identifiers to the same node (one for each ring) and hence several routing tables need to be updated. Our proposal keeps the same identifiers for both lower and upper ring, reducing the amount of required updates. We also use high-capacity devices for both contract replication and non-trivial matching computation.

5. CONCLUDING REMARKS

This paper presented a service discovery architecture for *heterogeneous* P2P systems. The key ingredient is a two-layers DHT, i.e., two overlapped Chord rings, the lower enclosing *low-capacity* and *high-capacity* devices, and the upper enclosing *high-capacity* devices only. Chord, which we selected for its widely-known efficiency (w.r.t. both in time and in space) in storing/retrieving data, is used in combination with the Hilbert SFC function, which allows us to hash service contracts taking into account different service attributes (e.g., type of service and QoS) and to handle *partially* specified queries.

While the work presented in this paper is in its early stages, it describes the first design of a service discovery architecture which suitably:

- spreads the discovery workload among *all* the available devices, while greatly exploiting, when available, high-capacity devices to back-up service contracts, to run non-trivial matching algorithms and to save low-capacity devices' resources,
- adapts to varying network configurations, e.g., a full low-capacity devices network, thus handling continual devices connections, disconnections, and failures,
- satisfies the (*enhanced*) clients' requests reducing the response time and the number of the exchanged messages. The *best* matching of service contracts w.r.t. the currently available devices is guaranteed.

The classical DHT searching mechanisms and the matching algorithms run by network devices are the two main elements that the service discovery architecture suitably combines to satisfy devices' requests. Thus, two interesting lines for future work are identified:

- to develop *lightweight* matching algorithms – yet capable of (partially) considering ontology/behaviour information – to be run by low-capacity devices, and
- to enhance the DHT matching, that is, to (partially) engage ontology/behaviour information to hash service contracts, thus improving the efficiency of the service discovery task.

Moreover, we are developing a proof-of-concept implementation of the service discovery architecture to experiment it in several network configurations (i.e., heterogeneous networks, and full low-capacity devices networks) on a large number of queries and service contracts.

6. REFERENCES

[1] *UPnP-Device Architecture-v1.0*. `http://www.upnp.org/resources/documents.asp`, 2006.

[2] *JXTA v2.0 Protocol Specification*. `https://jxta-spec.dev.java.net/JXTAProtocols.pdf`, 2007.

[3] P. Flocchini, A. Nayak, and M. Xie. Hybrid-chord: A Peer-to-Peer System Based on Chord. In *ICDCIT 2004, LNCS 3347*, pages 194–203. Springer, 2004.

[4] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, rfc 2608. 1999.

[5] J.K.Lawder and P.J.H.King. Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve. *ACM Sigmod Record*, 1(30):19–24, 2001.

[6] M. Li, E. Chen, and P. C. y Sheu. A Chord-Based Novel Mobile Peer-to-Peer File Sharing Protocol. In *APWeb 2006, LNCS 3841*, pages 806–811. Springer, 2006.

[7] Q. Lin, R. Rao, and M. Li. DWSDM: A Web Service Discovery Mechanism Based on a Distributed Hash Table. In *Proceedings of GCCW06*. IEEE Computer Society, 2006.

[8] J. Liu and H. Zhuge. A Semantic-based p2p resource organization model R-Chord. *ScienceDirect The journal of Systems and Software*, pages 1619–1631, 2006.

[9] Plastic IST STREP Project. D3.1: Middleware Specification and Architecture, 2007. `http://www.ist-plastic.org`.

[10] O. Sahin, C. Gerede, D. Agrawal, A. Abbadi, O. Ibarra, and J. Su. SPiDeR: P2P-Based Web Service Discovery. In B. Benatallah, F. Casati, and P. Traverso, editors, *ICSOC 2005, LNCS 3826*, pages 157–169. Springer, 2005.

[11] F. Sailhan and V. Issarny. Scalable Service Discovery for MANET. In *Proc. of the* 3^{rd} *IEEE Int. Conf. on Pervasive Computing and Communications (PerCom 2005)*, pages 235–244. IEEE Computer Society, 2005.

[12] C. Schimidth and M. Parashar. A Peer-to-Peer approach to Web Service Discovery. *World Wide Web. Internet and Web Information Systems*, 7(2):211–229, 2004.

[13] C. Schimidth and M. Parashar. Chapter 4: Peer-to-Peer Information Storage and Discovery Systems. In *Peer-to-Peer Computing, the evolution to a Distructive Technology*, 2005.

[14] SMEPP Consortium. D3.3: Concrete architecture of secure EP2P middleware services, 2008. `http://www.smepp.org`.

[15] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[16] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A scalable p2p infrastructure of registries for semantic publication and discovery of Web services. In *Inf Tech. and Management 6*, pages 17–39, 2005.

Digital Object Identifier: 10.4108/ICST.MOBIQUITOUS2008.3904
http://dx.doi.org/10.4108/ICST.MOBIQUITOUS2008.3904