The Journal of Systems and Software 82 (2009) 809–824

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Dynamic Web Service discovery architecture based on a novel peer based overlay network ☆

S. Sioutas [a,*], E. Sakkopoulos [b], Ch. Makris [b], B. Vassiliadis [c], A. Tsakalidis [b], P. Triantafillou [b]

[a] Department of Informatics, Ionian University, 7 Tsirigoti Square, 49100 Corfu, Greece
[b] Computer Engineering and Informatics Department, University of Patras, Greece
[c] Computer Science, Hellenic Open University, Patras, Greece

ARTICLE INFO

*Article history:*
Received 18 June 2008
Received in revised form 12 November 2008
Accepted 13 November 2008
Available online 3 December 2008

*Keywords:*
Web Services Discovery
Peer to Peer overlay networks
Databases

ABSTRACT

Service Oriented Computing and its most famous implementation technology Web Services (WS) are becoming an important enabler of networked business models. Discovery mechanisms are a critical factor to the overall utility of Web Services. So far, discovery mechanisms based on the UDDI standard rely on many centralized and area-specific directories, which poses information stress problems such as performance bottlenecks and fault tolerance. In this context, decentralized approaches based on Peer to Peer overlay networks have been proposed by many researchers as a solution. In this paper, we propose a new structured P2P overlay network infrastructure designed for Web Services Discovery. We present theoretical analysis backed up by experimental results, showing that the proposed solution outperforms popular decentralized infrastructures for web discovery, Chord (and some of its successors), BATON (and it's successor) and Skip-Graphs.



## 1. Introduction

The continuous expansion of the Internet and its relating technologies has created new marketing opportunities: traditional monolithic approaches in information system design are giving way to Service Oriented Computing (SOC). SOC supports the development of applications as if they were a connected network of functionalities (services) available, in a network-enabled environment, within and across different organizations (Singh and Huhns, 2005).

Web Services (WS), "a vision of loosely coupled interaction between components, programs, and applications" and a major implementation technology for SOC, is becoming a driver for business integration (Borenstein and Fox, 2004). Nevertheless, as more and more WS become available by many vendors, an old, search engine, problem is reappearing in a new form: searching (discovery) mechanisms of WS are not efficient both in response times and quality of results. Currently, the majority of Web Services are developed for internal enterprise use. Web Services will become more widely adopted in the years to come, allowing much broader intra- and inter-enterprise integration. It is anticipated that there will be an increasing requirement for automated service discovery, enabling further Web Services interaction with even less human effort. The 'information stress' problem in highly distributed environments is one of the most interesting and difficult for computer science, and has already attracted significant attention (Schmidt and Parashar, 2004; Yu et al.,2004a,b; Panagis et al., 2008; Diamadopoulou et al., 2008; Adamopoulou et al., 2007).

Web Service registries are helping to narrow down the negotiation and searching time needed for service discovery. Their basic concept lies in the matching mechanism of the contractual and technical profile of the query to that of the WS. In order for the searching procedure to be fast and more importantly automatic, information needs to be machine processable. Semantic Web technologies have already provided the first standards and tools towards machine to machine interaction; truly dynamic automatic discovery however is not yet in our grasp. WS technology is still categorised among most recent members of the Web Engineering area and it has already attracted both scientific community and vendor attention (Sycara et al., 2003). Promising steps have been made towards making WS a worldwide supported solution for application to application (A2A) interaction. Several recent technologies have been developed to support functionality in terms of service communication (e.g. SOAP), description (e.g. WSDL), and discovery (e.g. UDDI).

At present, Web Services are mainly advertised in catalogues which are based on the Universal Description, Discovery and Integration standard (UDDI). UDDI has become the predominant technological environment for WS Discovery. A large number of such centralized registry implementations for intra-enterprise communication, each focusing on registering services of interest

☆ Preliminary version of this paper was presented in Proceedings of IEEE International Conference on Information Technology: Coding and Computing, Next – Generation Web and Grid Systems (IEEE/ITCC 2005), pp. 193–198.

* Corresponding author.
*E-mail addresses:* sioutas@ionio.gr (S. Sioutas), sakkopul@ceid.upatras.gr (E. Sakkopoulos), makri@ceid.upatras.gr (Ch. Makris), bb@eap.gr (B. Vassiliadis), tsak@ceid.upatras.gr (A. Tsakalidis), peter@ceid.upatras.gr (P. Triantafillou).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.11.845

to respective groups, are anticipated. As a result, apart from the number of different Web Services to select from, there is also the challenge of dealing with a large number of specialized, sector/business-oriented registries during service discovery and selection. Centralisation may lead to significant performance bottlenecks (Ouzzani and Bouguettaya, 2004).

A solution to the bottleneck problem that is gaining popularity among researchers is decentralized infrastructures based on P2P (Peer to Peer). The convergence of WS and P2P computing promises some exciting results (Papazoglou et al., 2003). Most existing decentralized discovery solutions in practice are based on the Chord (Stoica et al., 2003; Zhang et al., 2003) and BATON (Jagadish et al., 2005, 2006) infrastructures as well as the best theoretical solution refers to Skip-Graph (Goodrich et al., 2006).

This paper, based on the ideas initially expressed in Sakkopoulos (2005), presents a novel efficient and fault tolerant discovery-search infrastructure for P2P Web Service Discovery, called NIPPERS (Network of InterPolated PeERS). The proposed solution provides support for processing: (a) exact-match queries of the form "given a key, locate the node containing the key" and (b) range queries of the form "given a key range, locate the node/nodes containing the keys that belong to this range". NIPPERS is an extensive upgrade of the Interpolation Search Tree Data Structure (Mehlorn and Tsakalidis, 1993) for distributed P2P environment. Results from our theoretical analysis show that the communication cost of the query and update operations scale double-logarithmically with the number of NIPPERS nodes, outperforming three of the most popular decentralized structures, Chord (and some of its successors), BATON (and it's successor) and Skip-Graphs.

Furthermore, the system is robust with respect to failures, providing a solution in this way for (1) a fully decentralized registry environment, (2) a decentralized mapper of WSDL description files and/or XML query files for WS to keys stored in the overlay network peers, and (3) Quality of WS requirements.

The rest of this paper is structured as follows. Section 2 overviews related work in Web Services, P2P Computing and P2P Web Service Discovery. Section 3 includes details about implementing XML Web Service Discovery on P2P overlay networks and introduces the NIPPERS architecture. Section 4 analyses the proposed solution functionality. Section 5 discusses fault tolerance and WS QoS issues contributing a technique that increases the robustness of NIPPERS to failures. In Sections 6 and 7, implementation issues and the experimental evaluation of the proposed solution are presented respectively. Finally, Section 7 outlines items for future work and conclusions.

## 2. Theoretical background

### 2.1. Web Service management and standards

Web Services management may be considered as a threefold case involving description, discovery and interaction issues. Description is mainly about interface and functionality: what are the characteristics of a WS and how it works in order to solve a specific problem. Description is based mainly on WSDL (Web Services Description Language). Discovery is about finding the right WS for the right task and interaction describes how different WS cooperate with each other to perform that task. These three procedures are strictly interconnected. For example, the description of a WS highly affects the discovery mechanism: semantically rich descriptions increase the quality of query results and thus discovery efficiency (Sycara et al., 2003).

Security is also an increasing concern of the WS community. SOC implementations which use combinations of WS face more threats than usual systems because they inherit the security risks of the autonomous entities involved. This is natural since negotiation and coordination activities between WS take place over the network. WS security can be based on open standards for supporting integrity, confidentiality and user authentication.

Standards and protocols are of great importance for the wider acceptance of WS by the industry. Most protocols are generally purposed to transit into open standards but they have not yet reached a state of maturity (Singh and Huhns, 2005). WS standardization is still a complex process. Many vendors have announced their own versions, versions that serve somewhat different visions of what WS are and how they must be used. This has led to a variety of competing, conflicting, or overlapping protocols promoted by different standardization bodies and vendors. Organisations such as W3C, OASIS, WS-I, and Liberty Alliance and vendors such as IBM, Microsoft, Sun, Oracle and BEA are promoting different solutions. Table 1 summarizes a categorisation of the main approaches for WS standardisation.

WS discovery in particular, is based on registries that make available (advertise) information concerning functionalities, interfaces, creator, pricing, etc. The structure of these registries, initially proposed by OASIS and later gained wider acceptation, is based on UDDI. UDDI adopts a centralized, client–server model: WS are registered in a UDDI registry and clients are able to search it in order to find the appropriate solution to their problem. This method is already proved to be problematic in terms of efficiency and uniformity (Li et al., 2004; Ouzzani and Bouguettaya, 2004). The client–server model suffers from performance bottlenecks when too many users search the same registry at the same time. The classic approach to add more servers or use load balancing techniques is not an efficient (or cost effective) solution. Decentralized approaches to WS discovery have already been proposed by large software vendors: Microsoft's WS-Inspection is a mechanism that relies on XML description in order to search web sites for available WS and accompanying usage rules (Singh and Huhns, 2005).

Furthermore, the ever-increasing need for quality of WS (QoWS) provisioning complicates even more selection procedures. QoWS may be seen from different perspectives such as basic service provision (e.g. network latency) or functionality. The latter refers directly to WS discovery since a query may return more than one results that meet the functional requirements but provide different quality of service attributes. This problem resembles that of search engine persuasion: lots of results- low quality. This

**Table 1**
Standards used for Web Services management.

| Operation | Standard |
|---|---|
| *Process* | |
| Publication and discovery | UDDI, USML, WS-Inspection |
| Orchestration | BPEL |
| Presentation | WSXL, WSRP |
| *Description* | |
| Definition | WSDL |
| User interface | WSUI |
| Business process design | XLANG |
| *Interaction* | |
| Messaging | SOAP |
| *Security* | |
| Representation of access control policies | XACML |
| Integrity and confidentiality | WS-Security (Web Services security language) |
| Legal | LegalXML |
| Negotiation | Web Services secure conversation language |
| Security policies | WS-SecurityPolicy |
| User authentication, attribute and authorisation assertion | SAML (security assertion markup language) |

**Table 2**
P2P vs. Web Services.

| Feature | P2P | WS |
|---|---|---|
| Peer client/server mode | Loose | Strict |
| Focus | Information | Activity |
| Decentralisation | High | Medium |
| Description/messaging | XML-based | XML-based |
| Security | Low | High |
| Peer connectivity time span | Short | Medium-Long |
| Architecture complexity | High | Medium |

problem is being addressed by semantically enriched WS, semantic Web Services. Semantic approaches using UDDI/DAML-S registries increase the quality of search results but do not avoid performance bottlenecks (Zou et al., 2004; Verma et al., 2005).

### 2.2. Peer to Peer oriented computing

Peer to Peer (P2P) is a relatively recent, highly distributed computing paradigm that enables sharing of resources and services through direct communication between peers (Androutsellis-Theotokis and Spinellis, 2004). Extending the traditional model where most computers on a network act as clients, P2P introduces the concept of the simultaneous client/server mode: peers act both as clients and as servers. This form of collaborative networking is based on dynamic and failure-tolerant architectures. Dynamicity refers to the ability of P2P networks to include/exclude peers at run time while maintaining an adequate level of performance. Failure (fault) tolerance refers to the ability to overcome connectivity problems posed by peer failure (Liben-Nowell et al., 2002).

The advantage of P2P technology is the ability to handle failures, manage transient peer populations and most importantly, to leverage SOC architectures. It should be noted that P2P computing is still in an evolving state and much needs to be done to overcome complex issues including security, network bandwidth, and architecture design. Androutsellis-Theotokis and Spinellis (2004) and Liben-Nowell et al. (2002) provide an excellent review for P2P technologies and protocols.

The convergence of P2P and WS has already been proposed by many researchers (Yu et al., 2004b; Papazoglou et al., 2003; Taylor and Harrison, 2005; Tsalgatidou and Koutrouli, 2005; Baresi et al., 2007; Sioutas et al., 2008; Li et al., 2007) since both approaches leverage the SOC and they may benefit from each other's strengths. It is often stated that difference is what makes convergence intriguing. Both approaches enable communication of loosely coupled systems and rely on open standards. Discovery is paramount for both approaches although P2P uses a more decentralized model than WS. XML is used by both to enable communication/messaging. Peers use a loose client/server mode while WS a stricter one in the sense that although a WS may publish or receive information, this is made under strict rules and conditions. It appears that there are more, although not yet matured, security standards on the development for WS than for P2P. Fault tolerance is the strong-point of P2P which focuses more on information transfer. On the other hand, WS focus on activity (information management/processing) and their strength is their flexibility and simple architecture. P2P architectures are, in some cases, more complicated and thus difficult to support.

Table 2 summarizes the similarities and differences of P2P and WS.

### 2.3. State of the art in P2P discovery of Web Services

Web Services themselves are rather new and the Web Service Discovery field is even newer. However, much attention has been given to this area (Li et al., 2004; Liu and Zhuge, 2005; Zou et al., 2004). An important trend in Web Service Discovery is to consider network nodes as peers which share information and are able to query other nodes. Discovery is based on P2P-based infrastructures, that is organisations of peers that do not constitute working applications, but provide a basis for WS elicitation. Decentralization granularity is an important in such infrastructures: when each Web Service is considered as a peer then the network is sparse, but when each registry is a peer, then the network is coarse (Schmidt and Parashar, 2004; Benatallah et al., 2005).

Decentralized WS discovery approaches use either a structured or unstructured P2P infrastructure (Liben-Nowell et al., 2002), although some hybrid solutions have appeared as well (e.g. super-peers). Unstructured P2P systems like Gnutella, MojoNation and Freenet define neighbours of peers in an ad-hoc manner and as such, they are appropriate for highly-transient peer populations communication (Clarke et al., 2000; Androutsellis-Theotokis and Spinellis, 2004). Due to the lack of any structure, location mechanisms face significant problems relating to availability, scalability and persistence. Nevertheless, peers enjoy a large degree of autonomy.

Structured infrastructures originated, partly, from the need to overcome the scalability problems of unstructured approaches. The basic idea is to use hashing in order to distribute a set of keys to each peer, each key corresponding to a specific resource. This approach greatly increases search efficiency when compared to unstructured P2P solutions but needs sophisticated load balancing mechanisms to sustain the proper structure (Koloniari and Pitoura, 2005).Current approaches mainly support exact-match queries and retain a quite good degree of scalability. Their main disadvantage is the increased difficulty to maintain the structure needed to efficiently communicate routing messages, especially in highly transient peer populations (Androutsellis-Theotokis and Spinellis, 2004). Most structured approaches are based on Distributed Hash Tables (DHTs). Overall, these systems provide efficient processing of location operations so that, given a query with an object key, they locate (route the query to) the peer node that stores the object. Some DHT-based systems are characterized as structured because, in general, they rely on lookups of a distributed hash table, a technique that imposes a virtual structure on the system emerging by the way that peers define their neighbours (Zhang et al., 2003). There are several P2P DHT architectures like Chord (Stoica et al., 2003), CAN (Ratnasamy et al., 2001), Pastry (Clarke et al., 2000; Rowstron and Druschel, 2001), Tapestry (Zhao et al., 2004), P-Grid (Aberer et al., 2003) and Kademlia (Mayamounkov and Mazieres, 2002). Sun's JXTA uses specialized peers called hubs that can register with other hubs as information providers, providing a middle ground between decentralized and centralized approaches.

CAN and Chord are the most commonly used infrastructures upon which many current and popular solutions are built. The extremely popular Chord P2P data lookup protocol is used as an overlay, consisting of Service Peers (SP). Each SP is mapped to several Logical Machines (Different Machines corresponding to the same hardware). Each Logical Machine maintains the necessary interfaces to map and search WSs in the P2P network. Chord-based solutions include Speed-R (Sivashanmugam et al., 2004), semantic enhancements to the basic structure (Liu and Zhuge, 2005), dimension reducing indexing schemes (Schmidt and Parashar, 2004), routing (Ganesan and Manku, 2004) and lookup latency (Zhang et al., 2003; Cordasco et al., 2004). Speed-R system (Sivashanmugam et al., 2004) is a P2P ontology based WS storage and retrieval system. Some nodes in the P2P subsystem are assigned registries, which in turn are partitioned according to their specific domain. Each domain is represented using an ontology. The speed-R P2P infrastructure is based on JXTA implementation. Its architecture is designed upon the assumption that each peer is assigned a role such as controlling updates and propagating them or accepting

new imported data. As a consequence, this system suffers from the single-point failure hazard.

Significant efforts for building new infrastructures for P2P WS discovery are P-tree (Crainiceanu et al., 2004), P-Grid (Aberer et al., 2003), BATON (Jagadish et al., 2005), BATON* (Jagadish et al., 2006) and Skip-Graph (Goodrich et al., 2006). The former is a distributed index P2P structure that supports ranges queries by storing parts of semi-independent B+-trees at each peer. The average cost per operation scales logarithmically to the number of peers. P-Grid is a scalable access structure based on a virtual distributed search tree. It uses randomized techniques to create and maintain the structure in order to provide decentralization. BATON is an overlay structure based on the binary balanced tree in which each peer in the network maintains a node of the tree. The worst-case cost per operation scales logarithmically to the number of peers. BATON* is an overlay multi-way tree structure based on B-tree with better searching performance. The penalty paid is a little larger update cost. Skip-Graph is a multiple (multilayer) list. It uses randomization in order to create and maintain the distributed structure.

## 3. A new index for Web Service Discovery

### 3.1. Web Service Discovery roadmap: the basic steps

NIPPERS has a sparse network granularity with node acting as a Web Service peer. This configuration not only enables WS consumption, but also maps WS to peers in the P2P overlay network. Therefore the Web Service providers act (a) as the nodes of the NIPPERS overlay network beside (b) delivering the WS functionality itself.

When using P2P overlay networks for WS discovery, then the WS selection mechanisms basically are about finding how to route the queries to appropriate peers, that can facilitate service consumption requests. In NIPPERS, similar to the registry paradigm, a WS request can be described in XML. This is in accordance with the WSDL descriptions stored in the WS node peers.

Web Services can be described in different ways, according to existing standards (e.g. WSDL), and can be characterized by a set of keywords. We use these keywords to index the Web Service description files, and store the index at peers in the P2P system.

In particular, NIPPERS queries allow keyword-matching queries on service name/abstract, service category and tModel (see query samples below). These are the most common features currently used with discovery directories (e.g. UDDI) to characterize a service. This set of query features is also trivially extendible to support future interesting features (e.g. QoS) of the service (see multi-attribute query sample and Section 4.1 for multi-attribute query details).

In this way every query for Web Service in the NIPPERS P2P can be conceived to be either in a simple case a single-attribute (i.e. keyword) one or in general a multi-attribute one (i.e. multiple keywords).

As a consequence in the general case, a set of keywords describes the WS and forms an ordered $d$-tuple where each attribute/keyword is finally mapped into a key. In this way we can specifically define the following types of query (and matching).

Let $S$ be a collection of $N$ points, each of which is an ordered $d$-tuple $r_1, \ldots, r_d$ of values $r_i$, $1 \leqslant i \leqslant d$. Each component of the $d$-tuple is called an attribute or a key. A retrieval request is the specification of certain conditions that must be satisfied by the keys of the retrieved records. The queries considered, are categorised as follows:

- Range query specifies d ranges, one for each key.
- Partial range query specifies $s < d$ key ranges, with the remaining $d–s$ unspecified.
- Exact-match query specifies an exact value for each key.
- Partial match query specifies $s < d$ key values, with the remaining $d–s$ ones unspecified.

The last two types of queries can are special cases of the general range query, if we allow the exact range $[x, x]$ and the infinite range $(-\infty, +\infty)$, as a possible query range for each key. As a result, an exact-match query can be represented as a general range query, in which all d ranges are exact. In the same manner, a partial match query has only s exact ranges.

In the figures below NIPPERS queries allow keyword matching on service name/abstract or multi-attribute queries with keyword matching on service name, service category, tModel and a QoS characteristic in this sample "Price-per-transaction". These are the most common features currently used with discovery business registries (e.g. UDDI) to characterize a service.

*Simple Query SOAP Message for Web Service Search at Nippers*

```
POST /NIPPERS.asmx HTTP/1.1
Host: morfeas.ceid.upatras.gr
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://morfeas.ceid.upatras.gr/
  NIPPERSSearch
<?xml version = "1.0" encoding = "utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xmlns:xsd="http://www.w3.org/
  2001/XMLSchema" xmlns:soap="http://
  www.schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <NIPPERSSearch xmlns="http://
  www.morfeas.ceid.upatras.gr/">
    <ServiceName>Rent-A-Car Service</ServiceName>
  </NIPPERSSearch>
</soap:Body>
</soap:Envelope>
```

*Multi-attribute Query SOAP Message for Web Service Search at Nippers*

```
POST /NIPPERS.asmx HTTP/1.1
Host: morfeas.ceid.upatras.gr
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://morfeas.ceid.upatras.gr/
  NIPPERSSearch
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance" xmlns:xsd="http://
  www.w3.org/2001/XMLSchema" xmlns:soap="http://
  www.schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <NIPPERSSearch xmlns="http://
  morfeas.ceid.upatras.gr/">
    <ServiceName>Rent-A-Car Service</ServiceName>
    <ServiceCategory>Tourist Services</
  ServiceCategory>
    <tModel>RentalInterfaceInOut</tModel>
      <QoSPricePerTransaction> 0.01 </
  QoSPricePerTransaction>
  </NIPPERSSearch>
</soap:Body>
</soap:Envelope>
```

Publishing the WS description upon the overlay network follows a set of similar steps.

(a) Step (a) includes parsing the service description metadata from the submitted XML Query that describes the functional (methods, input and output parameters, etc.) and non-functional (QoS characteristics) metadata of the Web Service one is looking for.
(b) Step (b) is to distribute the request keys in order to initiate the querying process.
(c) Step (c) is the NIPPERS discovery approach that is analysed further in Fig. 1.
(d) At step (d) the WS is discovered and therefore the requesting client can link and download the WSDL description file and/or bind upon the WS to consume its functionality.

Step (a) is broken into a number of more delicate processes in order to transform the XML query for a WS or the WSDL publishing on the overlay network. The central pre-processing step (a) consists of the following processes:

a.i. An XML parser is initialized to analyse the submitted request or WSDL description file into its node names and node values. Only informative nodes/metadata are kept such as WSDL description, WSDL keywords, WSDL business entity description, etc.
a.ii. The informative nodes are resolved into a set of descriptive keywords.
a.iii. Each service description request or publish is mapped upon key(s). In fact, the specific keyword(s) extracted (i.e. WSDL attributes) are mapped to key(s).

Step (b) represents the initiation of a key searching process for the request key, which is the key that maps the Web Service description request (in particular its keywords). In this way the discovery and selection problem is transformed into a key searching or insertion task. As a consequence the focus of this paper is to present a specifically fine-tuned and implemented P2P overlay infrastructure analysed in Section 3.3. The approach presented above is typical and implemented by a number of P2P WS discovery solutions (Schmidt and Parashar, 2004; Li et al., 2004; Zou et al., 2004) and it is not within the scope of this paper to further analyze it.

Other DHT-based routing and locating algorithms, like Chord, support only exact-match queries. A solution is to provide some extensions for XML based conditional match support. In our solution, we have taken into consideration such cases. In addition, NIPPERS supports partial matching. This kind of matching is particularly useful in order to provide reassurance for quality of WS selection (see Section 5 for details).

### 3.2. Background information about interpolation searching – the IST

The idea for the adaptation of an IST is the sub-logarithmic O(log log $n$) complexities that achieves as a centralized structure. In this paper we deliver a decentralized overlay that still achieves similar complexities. We apply the same logic as IST but not directly to the keys, but rather to the peer-nodes and that is why we call this solution a Network of Interpolated Peers. Furthermore in this section we provide short background information about interpolation searching using the IST for complicity. Readers who already have knowledge of IS are urged to continue next at Section 3.3 for the NIPPERS infrastructure analysis.

Improvements regarding the search complexities can be obtained if certain classes of input distributions are considered. A notorious example is the method of *interpolation searching* which, for random data generated according to the *uniform* distribution achieves $\Theta(\log\log n)$ expected search time (Gonnet et al., 1980). Willard in Willard (1985) showed that this time bound holds for an extended class of distributions, called regular. A density $\mu$ is regular if there are constants $b_1, b_2, b_3, b_4$ such that $\mu = 0$ for $x < b_1$ or $x > b_2$, and $\mu(x) \geqslant b_3 > 0$ and $|\mu'(x)| \leqslant b_4$ for $b_1 \leqslant x \leqslant b_2$.

A natural extension is to adapt interpolation search into dynamic data structures, that is, data structures which support

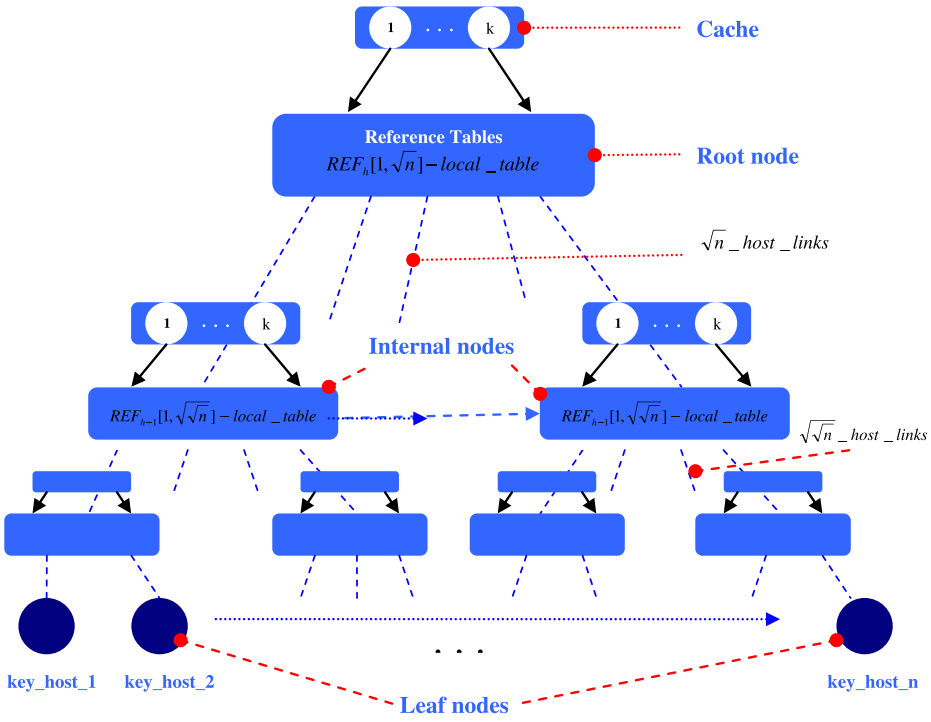


**Fig. 1.** The NIPPERS P2P infrastructure.

insertion and deletion of elements in addition to interpolation search. Their study was started with research focused on insertions and deletions performed according to the uniform distribution, and continued for – random insertions and deletions, where $\mu$ is a so-called smooth density (Andersson and Mattson, 1993; Mehlorn and Tsakalidis, 1993). An insertion is $\mu$-random if the key to be inserted is drawn randomly with a density function $\mu$; a deletion is random if every key present in the data structure is equally likely to be deleted.

The notion of smooth input distributions that determine insertions of elements in the update sequence were introduced in Mehlorn and Tsakalidis (1993), and were further generalized and refined in Andersson and Mattson (1993). Given two functions $f_1$ and $f_2$, a density $\mu = \mu[\alpha, b](x)$ is $(f_1, f_2)$-smooth if there exists a constant $\beta$, such that for all $c_1$, $c_2$, $c_3$, $\alpha \leqslant c_1 < c_2 < c_3 \leqslant b$, and for all integers $n$, it holds:

$$\int_{c_2-\frac{c_3-c_1}{f_1(n)}}^{c_2} \mu[c_1, c_3](x)dx \leqslant \frac{\beta f_2(n)}{n}.$$

The class of smooth distributions is a superset of regular, normal, uniform as well as of real world skew distributions like weibull, zipfian, binomial or power law (for details see Kaporis et al., 2006).

In (Mehlorn and Tsakalidis, 1993), a dynamic interpolation search data structure was introduced, called Interpolation Search Tree (IST). This data structure requires $O(n)$ space for storing $n$ elements. The amortized insertion and deletion cost is $O(\log n)$, while the expected amortized insertion and deletion cost is $O(\log\log n)$. The worst-case search time is $O(\log^2 n)$, while the expected search time is $O(\log\log n)$ on sets generated by $\mu$-random insertions and random deletions, where $\mu$ is $(\lceil n^\alpha \rceil, \sqrt{n})$-smooth density function and $1/2 \leqslant a < 1$. An IST is a multi-way tree, where the degree of a node $u$ depends on the number of leaves of the subtree rooted at $u$ (in the ideal case, the degree of $u$ is the square root of this number). Each node of the tree is associated with two arrays: a REP array which stores a set of sample elements, one element from each subtree, and an ID array that stores a set of sample elements approximating the inverse distribution function. In particular, for a node with degree $\sqrt{m}$ and having m leaves in its subtree, the ID array divides the interval covered by the node in $m^a$ subintervals, where $1/2 \leqslant a < 1$. Each interval is associated with a pointer to the proper subtree. The search algorithm for the IST uses the ID array in each visited node to interpolate REP and locate the element, and consequently the subtree where the search is to be continued.

Andersson and Mattson (1993) explored further the idea of dynamic interpolation search by observing that: (i) the larger the ID array, the bigger becomes the class of input distributions that can be efficiently handled with an IST-like construction; and (ii) the IST update algorithms may be simplified by the use of a static, implicit search tree whose leaves are associated with binary search trees and by applying the incremental global rebuilding technique of Overmars and Leeuwen (1981).

The resulting new data structure in Andersson and Mattson (1993) is called the Augmented Sampled Forest (ASF). Assuming that $H(n)$ is an increasing function denoting the height of the static implicit tree, Andersson and Mattson (1993) showed that an expected search and update time of $\Theta(H(n))$ can be achieved, when the input is $(n^* g(H(n)), H^{-1}(H(n)-1))$-smooth, where $H(n)$ is an increasing function representing the height of their tree structure and $g$ is a function satisfying:

$$\sum_{i=1}^{\infty} g(i) = \Theta(1).$$

In particular, for $H(n) = \Theta(\log\log n)$ and $g(x) = x^{-(1+\varepsilon)}$ $(\varepsilon > 0)$, they get $\Theta(\log\log n)$ expected search and update time for any $(n/(\log\log n)^{1+\varepsilon}, n^{1-\delta})$-smooth density, where $\varepsilon > 0$ and $0 < \delta < 1$ (note that $(\lceil n^a \rceil, \sqrt{n})$-smooth $\subset (n/(\log\log n)^{1+\varepsilon}, n^{1-\delta})$-smooth). The worst-case search and update time is $O(\log n)$, while the worst-case update time can be reduced to $O(1)$ if the update position is given by a finger. Moreover, for several but more restricted than the above, smooth densities, they can achieve $O(\log\log n)$ expected search and update time complexities; in particular, for the uniform and any bounded distribution the expected search and update time becomes $O(1)$.

These are the best results so far in both the realm of dynamic interpolation structures and the realm of dynamic search tree data structures for $\mu$-random insertions and random deletions on the RAM model.

### 3.3. The NIPPERS infrastructure

NIPPERS provides an exponential distributed tree-like structure upon which key-based searching can be performed. In terms of bandwidth usage, searching scales very well since no broadcasting or other bandwidth consuming activities take place. Since all searches are key based, there are two possibilities: either (a) each host implements the same algorithm that translates keywords/attributes queried to binary keys or (b) another service provides the binary key (or keys). This service accepts keyword based queries and can respond with the corresponding key. The second approach is more precise. It is also possible to use a more centralized implementation for such a service. For the rest of the paper, we assume that the key is available. In the following sections we describe an algorithm for the first case. We also suppose that the set of keys on each host retain a total order.

The NIPPERS search tree is an implicit virtual Interpolation Search Tree $T$ (Fig. 1). More specifically, the structure consists of two levels with the top level being a static interpolation search tree, and the bottom level being a family of buckets. The structure is maintained by incrementally performing global reconstructions. Let $n_0$ be the number of elements stored at the time of the latest reconstruction. After that time when the number of updates exceeds $rn_0$, where $r$ is an arbitrary constant, then the whole data structure is reconstructed. Let $n$ be the number of stored elements at this time. After the reconstruction, the number of buckets is equal to $n/\ln n$, while each bucket contains $\Theta(\ln n)$ elements. During the subsequent insertions/deletions, the top level remains intact while only the buckets are affected. Note that during these update operations; it is not at all obvious how to bound the size of the buckets. In (Kaporis et al., 2003), an idea of general scientific interest was presented: By ordering the initial $K$ keys, choosing then as key-host's (leaf_peers's) representatives the 1st key, the $\ln n$st key, the $2\ln n$st key, etc. and modelling the insertions/deletions as a combinatorial game of bins and balls, the size of each bucket is expected w.h.p. polylog($n$), for elements that are drawn from a $\mu(\cdot)$ unknown distribution (Mehlorn and Tsakalidis, 1993).

Next, we will show how this data structure can be virtually imposed on a P2P network. Let $p_i$ denote a peer node. We distinguish between leaf_peers and node_peers: If $p_i$ is a leaf_peer then it maintains a set $S_i$ of ordered $k$-bit binary keys $k_i = b_1 \cdots b_k$, where $k$ is less than or equal to $n_1$, for some bounded constant $n_1$ which is the same for all $p_i$. This ordered set of keys denotes the key space that the peer is responsible for and is equivalent to the set of elements stored in a bucket of the virtual structure.

If $p_i$ is a node_peer of the NIPPERS network, it is then associated with an internal node of the top level of the virtual data structure. So it contains two tables: a local table REF of sample elements (one for each of its subtrees) and an ID array. By using the ID array it is possible to interpolate the REF table to determine the subtree in which the search procedure will continue.

Let leaf_peer$_1$, leaf_peer$_2$,…,leaf_peer$_n$ be the leaf peers ordered according to their number. Then, the key-sets $S_j$ retain a global order. This means that

$$\forall S_j, S_m, \quad 1 \le j \le n, \quad 1 \le m \le nj \neq m, \quad \text{if } \min\{S_j\} < \min\{S_m\} \quad \text{then } \max\{S_j\} < \min\{S_m\}.$$

Thereupon, we are sorting the key-sets above providing a leaf oriented distributed data structure (see Fig. 1). Finally, for each node we explicitly maintain parent, child, and sibling pointers. Pointers to sibling nodes will be alternatively referred to as level-links. The required pointer information can be easily incorporated in the construction of the NIPPERS search tree.

## 4. Analysis of basic functionalities

### 4.1. Search algorithm analysis

**Theorem 1.** *Assume a NIPPERS search tree with parameters* $R(s_0) = (s_0)^{1-1/\delta}$, *where* $0 < \delta < 1$ and $s_0 = n$. *Then, exact-match queries require* $O(\log\log n)$ *expected number of hops for elements that are drawn from a* $\mu(\cdot)$ *unknown distribution* (Kaporis et al., 2003).

**Proof.** Assume that a peer performs a search for the key $k$. To perform the search, a connection to a peer $p$ in the NIPPERS network is established and the call NIPPERS_search($p$, $k$) is performed: we walk towards the root, say we reached node $u$. We check whether $k$ is a descendant of $u$ or $u$'s right neighbour on the same level by searching the REF table of $u$ or $u$'s right neighbour respectively. If not, then we proceed to $u$'s father. Otherwise we turn around and search for $k$ in the ordinary way. Suppose that we turn around at node $w$ of height $h$. Let $v$ be that son of $w$ that is on the path to the peer $p$. Then all descendants of $v$'s right neighbour lay between the peer $p$ and the key $k$. The subtree rooted at $v$'s right neighbour is a NIPPERS tree for $n' \leqslant n$ elements, and its height is $h = \Theta(\log\log n)$. So, it follows that the time complexity is such as stated in Theorem 1. □

**Remark 1.** According to advantages of smooth distributions (Andersson and Mattson, 1993; Mehlorn and Tsakalidis, 1993; Kaporis et al., 2003, the use of ID table speeds up in O(1) expected time with high probability the searching procedure at each peer. But this may never be occurred if the elements are drawn from an $\mu(\cdot)$ unknown distribution. Considering now that in P2P model the processing cost at each peer in comparison with communication overhead (messages) between peers is negligible, ID tables, which interpolate REF tables, are really redundant. REF tables only are enough. As a consequence, our basic result holds not only for smooth but for any $\mu(\cdot)$ unknown distribution according to which we structured and bounded the load of each P2P on the algorithmic game presented in Kaporis et al. (2003).

**Remark 2.** Exploiting the order between the key_sets on the leaves then Range Queries of the form $[k_l, k_r]$ require $O(\log\log n + |A|)$ hops, where $|A|$ is the number of leaf peers between the peers responsible for $k_l$, $k_r$ respectively.

**Remark 3.** We can extend our structure to Multi-dimensional Space, by using the flexible method proposed in Jagadish et al. (2006) for supporting multi-attribute queries. In particular, we divide the whole range of attributes into several sections: each section is used to index an attribute (if it frequently appears in queries) or a group of attributes (if these attributes rarely appear in queries). Since NIPPERS can only support queries over one-dimensional data, if we index a group of attributes, we have to convert their values into one-dimensional values (by choosing Hilbert

**Table 3**
Pseudo-code for the search algorithm.

```
Node p *NIPPERS_search (p, )
{
  int j;
  bool move_right=false;

  if ( p is responsible for this k)
      return p;
  if (someone else is responsible)
      check whether k is to the left or right of p;
  \\ say k is to the right of p\\
  p_next=father(p)
  S_0 = |T_{p_next}|
  While (k > REF_{p_next}[(sqrt(S_0))] && move_right = false)
    {
      p'=right_sibling of p_next;
      S' = |T_{p'}|;
      if (k <= REF_{p'}[(sqrt(S'))])
        {
          p_next= p';
          move_right=true;
        }
      else
          p_next=father (p_next);
      S_0 = |T_{p_next}|
      host = send_search(p_next, k);
    }
  While ( p_next is not a key_host)
    {
      j=search (k,p_next);
      \\ Where search (key, node) denotes the procedure which
         returns an integer position j indicating the appropriate
         descendant where searching must continue\\
      p_next=&REF[j];

      host = send_search(p_next, k);
    }
  p=p_next;
  return p;
}
```

**Table 4**
Pseudo-code for the range-search algorithm.

```
NIPPERS_range-search (p, k_ℓ, k_r)
{
  p=NIPPERS_search (p, k_ℓ);
  Retrieve from leaf_peer p;
  p_next=Right Sibling (p);
  p=p_next;
While (p is left of leaf_peer k_r)
    {
      Retrieve from leaf_peer p;
      p_next=Right Sibling (p);
      p=p_next;
    }
}
```

Space Filling Curve or other similar methods). For example, if we have a system with 10 attributes: $a_1$, $a_2$,…,$a_{10}$ in which only 4 attributes from $a_1$ to $a_4$ are frequently queried (i.e. 90% of all queries), we can build 4 separate indexes for them. The remaining attributes can be divided equally into two groups to index, three attributes in each group. In this way, the number of replications can be significantly reduced from 10 down to 6.

Table 3 presents the pseudo-code for the search algorithm in NIPPERS.

Table 4 presents the pseudo-code for the range-search algorithm in NIPPERS.

### 4.2. Insertion/joining and deletion/leaving of elements/nodes – local reconstruction and load balancing

When we want to insert/delete a key from the structure we initially search for the node (key_host_peer) that is responsible for it (using a number of $O(\log\log n)$ hops) and then we simply insert/delete it from the appropriate node. After $O(n)$ insertions/deletions we perform, as already mentioned, a global reconstruction. However there are some cases when a leaf_peer overflows (or underflows). In the first case, we must join nearby a new leaf_peer. In the second case, the leaf_peer is leaving by moving first the few remaining keys to the left or right neighbours. We chose to deal with these joining/leaving operations by employing a series of local reconstructions that are implemented using appropriate counters $C(\ )$ attached to nodes. The local reconstructions can similarly handle element insertions and deletions (in a vein similar to Mehlorn and Tsakalidis (1993)) thus making global reconstruction obsolete. The transfer and storing of keys in the neighbour nodes as well as the local reconstructions achieve local load balancing.

By employing this series of local reconstructions it can be proved (the proof is omitted since it is quite similar to (Liu and Zhuge, 2005) that insertions and deletions require $O(\log\log n)$

**Table 5**
Pseudo-code for the insertion/joining operation.

```
Procedure JOINING_leaf_peer
{
   \\ Let T be the existing NIPPERS tree and v0,v1,…..,vk be the path
   from the new joined node v0 to the root vk\\
  Insert the overflowed key in v0;
 C(v0)=0;
 isize(v0)=1;
   \\ isize(v) denotes the initial size (weight) of Tv subtree the last time
   counter C(v) was set to zero \\
  For (i=1; i≤k; ++i) C(vi)=C(vi)+1;
     Find the maximum i such as: C(vi) ≤isize(vi)/4;
  If (there exists such i)
     Rebuild(T_vi);
}
```

**Table 6**
Pseudo-code for the deletion/leaving operation.

```
Procedure LEAVING_leaf_peer
{
     \\ Let T be the existing NIPPERS tree and v0,v1,…..,vk be the path
     from the leaving node v0 to the root vk\\
   Transfer the few remaining keys of v0 to the left or right neighbours;
   for (i=1; i≤k; ++i) C(vi)=C(vi)+1;
      Find the maximum i such as: C(vi) ≤isize(vi)/4;
   If (there exists such i)
      Rebuild(T_vi);
}
```

**Table 7**
Pseudo-code for the rebuilding operation.

```
Procedure Rebuild(Tv)
{
  Build a new NIPPERS structure for Tv;
  ∀ v∈T Set C(v)=0;
  ∀ v∈T define isize(v);
}
```

amortized expected number of hops. Tables 5–7 present the pseudo-code for the insert/joining, delete/leaving and rebuild (or overall load balancing) operations respectively.

Based on the basic result of Kaporis et al. (2003), if we generate the keys according to $\mu(\cdot)$ unknown distribution and chose the peer representatives according to the combinatorial game of balls into bins presented in Kaporis et al. (2003), then we can assure with high probability that the expected load of each peer never exceeds polylog$n$ size and never becomes zero. The latter means that in expected case with high probability split or delete operations will never be occurred. In other words, the re-organization of the whole p2p structure with high probability will never be occurred.

Summarizing for load balancing, we use the weight-balancing properties of counters on the path of root to the new inserted (joined) or old deleted (left) node and execute rebuildings (keep in mind that these operations w.h.p. will never occur) in order to retain the efficiency of all operations even in amortized case.

## 5. Fault tolerance and quality of WS: one country-side with local farms

The NIPPERS P2P overlay network for Web Services is a large P2P WS registry "country-side" consisted of a number of local WS registry "farms". The intuition behind the notion of the registry "farms" and the overall "forest" comes to deal with two problems initially:

(1) Possible bottlenecks and failures due to the existence of a centralized registry.
(2) Possible bottlenecks and failures due to the existence of a single centralized mapper of WSDL description files to keys as in the case of a traditional centralized Web Service registry. Please note that such a function is the most costly in a real WS scenario, as it needs to operate on the WSDL/semantics of the service.

A formal analysis of the above idea will be discussed in the following. In general, to provide for better balancing and tolerance to failures, a replication protocol is employed and upon a service providing peer node failure. The system finds a nearby node that stores a copy of the desired data. In our structure, for each node, we make $k$ redundant nodes with each of them storing a replicated copy of a data item, where $k > 1$ is a small positive constant. We make the assumption that the network is "$k$-robust", meaning that the simultaneous failure of all these nodes is impossible, thus, at least one node is alive in the network.

We also use the idea of topological partition on a graph network $G$ (Fig. 2) in order to ensure that node fails will affect the local area only. We want to design a network, where the neighbour_peer Web Service will be a real (physically) geographical neighbour and not a virtual_neighbour. Thus, a spatial decomposition is required. We divide the graph $G$ into an exponential number ($n^{\alpha}$, $0 < \alpha < 1$) of subgraphs $G_i$, $1 \leqslant i \leqslant n^{\alpha}$. For reasons of simplicity, we set $\alpha = 1/2$. Each of the subgraphs $G_i$ has $k$*-representative* nodes, where $k > 1$ a small positive constant. We suppose that each of the subgraphs is $k$-robust, that means the simultaneous shutdown of all representative nodes is impossible, thus at least one of the $k$-*nodes* is active on the network.

We partition each subgraph recursively until we find a small topological subgraph with no more than $c$ nodes, where $c > 1$, a small positive constant. As a result, a Distributed Exponential Structure is created (Fig. 3). Each node represents a subgraph and retains host-links to subgraphs that have been partitioned. The number of host-links is exponentially decreased while we get downwards in the tree. The host-links are created between the representative nodes of each subgraph. Moreover, according to searching algorithm of Table 3 previously described, every lookup query at source node $s$ requires the visiting of the two spines only of the subtree $T(w)$, where $w$ is the nearest common ancestor of source node $s$ and destination node $t$. Since $s$ is randomly selected

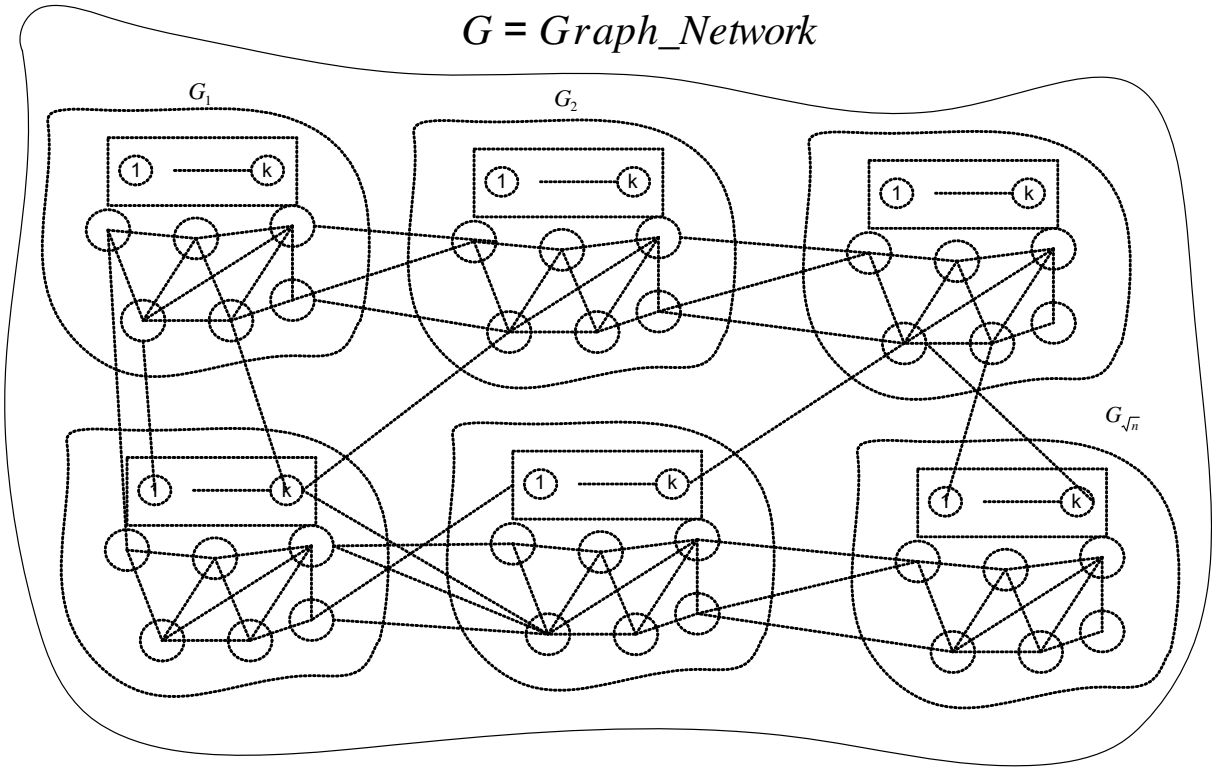


**Fig. 2.** Network graph partition.

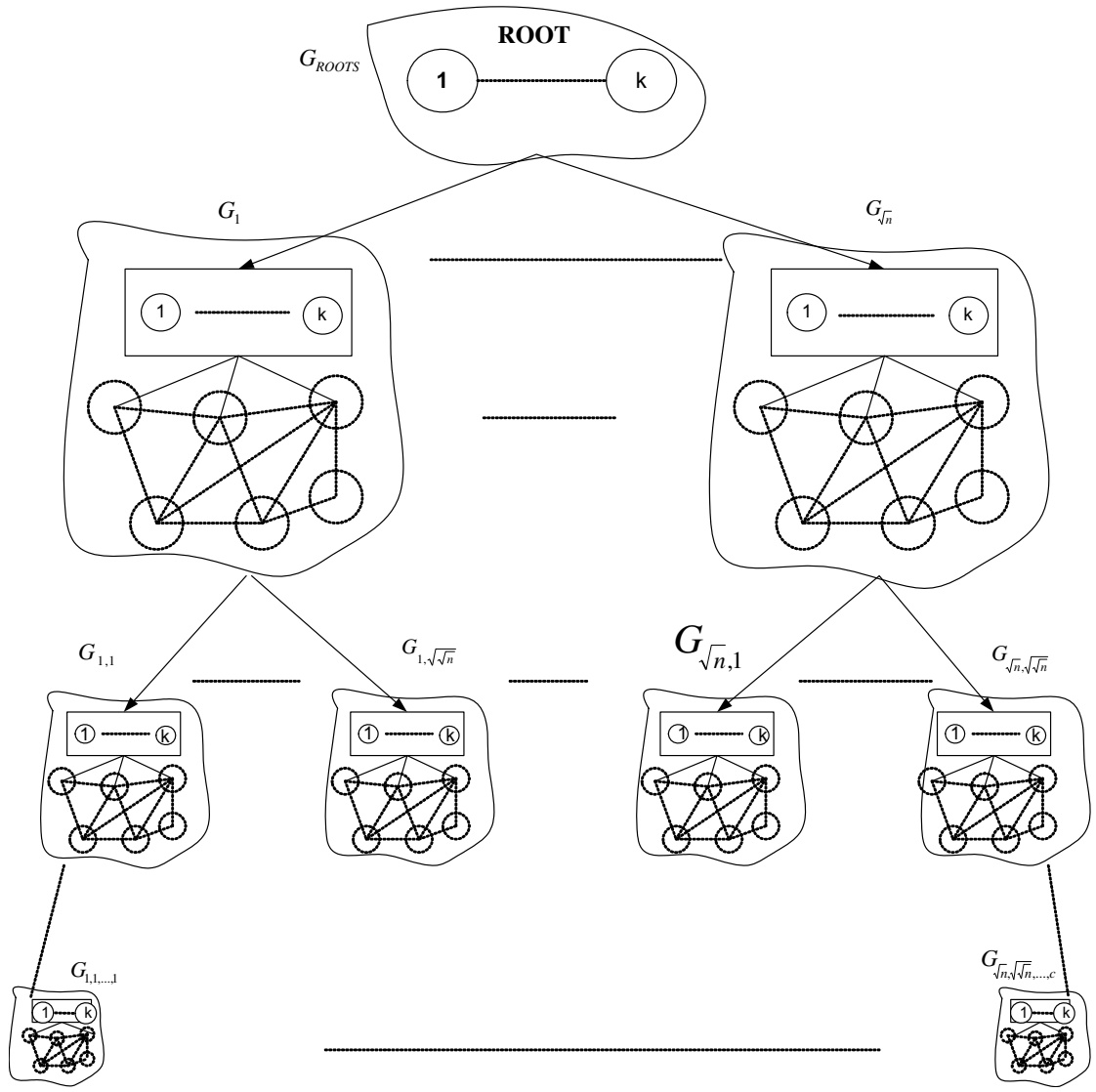


**Fig. 3.** The NIPPERS architecture.

during the lookup queries, the node paths are also randomly visited and as a consequence none node constitutes a hot spot. Obviously, level-links help to this purpose.

The next step is to store the ordered key_sets in the leaves of the topological structure above. The structure is key_based, so the key information must be stored at all nodes until the root. We perform the search, join and leave operations in the same way.

Since node fails affect the local area only, NIPPERS is highly fault tolerant. In the contrary, in BATON* system if $f$ is the number of failed nodes in the system, then the maximum number of nodes, which may be separated from the network as a result of the failure is from $f/(m+1)$ to $f/(m-1)$. For large m this ratio is very small, but for small m this ratio is very large.

Furthermore, the Quality of WS Provisioning is also a concern which is at least neglected in most of WS area work. As a result, neither a crystal definition nor globally accepted perceptions of QoWS exist. However, recent work, presented in Ouzzani and Bouguettaya (2004) and Ran (2003), breaks new ground in an attempt to define some of the QoWS parameters and methods of delivery. Below, we highlight the main parameters that set the basis of QoWS. *First comes the computational behaviour*, which involves execution attributes (e.g. network latency, throughput, etc.), security (encryption, authentication, etc.), privacy (enforcement of privacy policies, etc.) and availability (the probability of the service being available). An early approach appears in Makris et al. (2004) where a proposed discovery approach adaptively selects the most efficient WS among possible different alternatives with real-time optimized countable factors-parameters.

*Business behaviour* has much to do with QoWS in terms of execution cost and reliability of the service provider. Finally, there are *metadata constraints* that have to be followed regarding UDDI/WSDL parameters such as location, specific companies, etc. Using NIPPERS, range queries can retrieve a series of different Model descriptions. In this way, an environment for extra QoWS characteristics control is available where extensive QoWS validation can be performed in order to pick the final most efficient WS selection.

Finally in terms of space consumption NIPPERS behaves as following: Space(Nippers) = O($N$), in particular:

$$\begin{aligned} \mathrm{S(NIPPERS)} &= \mathrm{O}\left(\sqrt{N}+\sqrt{N}\cdot\sqrt{\sqrt{N}}+[\sqrt{N}\cdot\sqrt{\sqrt{N}}]\cdot\sqrt{\sqrt{\sqrt{N}}}+\cdots\right) \\ &= \mathrm{O}(N), \end{aligned}$$

since the routing table of each node located at level $i$ is an exponentially decreasing function of $i$, in particular $n^{\frac{1}{2^i}}$.

Furthermore, S(Skip-Graphs) = O($N$) and S(BATON) = S(BATON$^*$) = $N^*\log(N)$, since each node has a routing table of O(log$N$) size.

As a consequence there is no duplication of space, and Nippers can be directly compared to the space optimal structure of Skip-Graph.

## 6. Implementation issues

To evaluate the algorithms presented, a prototype is implemented. It is based on MS .NET technology framework for both the P2P service as well as the Web Service infrastructure (Ballinger, 2003). Development took place in Pentium 4 3 GHz computers with 512 MB RAM on Windows OS using Visual Studio .NET environment and C# language. Our choice to use the Microsoft .NET Framework is based on the fact that wide previous experience in the development tools existed. Generality is not lost though as other WS supporting technologies may be used such as J2EE.

The prototype uses the interface of an primer P2P sample provided in early provisions of .NET framework (beta versions) through the MSDN magazine which we have radically re-implemented using new utilities of the latest framework version to incorporate the proposed algorithms. It is a Windows .NET application that allows sharing Web Services information among service peers. The application has a Windows user interface, which contains three tabs that are used for managing, searching, and fetching WSDL files. The Search tab is used for searching the service for files and starting up file transfers. The Library tab provides a basic list of the files currently being shared. The Options tab provides access to settings information.

The prototype utilizes the .NET System.Web.Service namespace to publish Web Services. To represent the peer application, the Win Forms application model is used. Finally, ShareBaby follows the System.Net namespace to enable content transfers from one peer to another.

The prototype is developed for experimental use in a laboratory LAN environment and has not been tested over slower connections. Over a slower connection, the prototype might leave "orphaned" peers that will be unable to un-register files at shutdown. To overcome this obstacle, a time obsolescence mechanism for the least recently used peers can be activated to automatically de-register inactive peers.

The System.Web.Service namespace provides the necessary classes to consume and expose a WS. With the .NET Framework, creating a Web Service is as simple as creating a class in a regular server-side web page. Likewise, consuming this Web Service from the peer application is extremely easy and is accomplished by calling a method on a proxy class that is generated with Visual Studio. NET or with the WebServiceUtil.exe program in the .NET Framework SDK. For more general details on how Web Services work with .NET, see (Microsoft Visual Studio).

In this case client/server technology is mixed together with P2P concepts to create an application that is fairly simple due to the traditional nature of client/server, yet extremely powerful because it is able to use the resources of all peers registered on the network.

Fig. 4 depicts snapshots of the P2P application. One may recognize WS description based full text search functionality through this simple interface. In the part (A) of the figure, two clients have been connected on the NIPPERS network and they have been used to display the full list of the available Web Services in the catalogue. In part (B), a single client-interface is presented. The interface provides some basic information about each service matching the search criteria such as URI path, connection type (LAN, WAN, Unknown), IP address, network response time (average) and more according to the options selected. One may download the URI and WSDL of a service to examine it. Furthermore, the NIPPERS network catalogue list is exposed-published available for developers through a Web Service that allows all functionality that client-interface has in order to facilitate automation in Web Service consumption.

## 7. Evaluation and outline of contributions

### 7.1. NIPPERS vs. Chord, Skip-Graphs and BATON

For comparison purposes, an elementary operation's evaluation is presented in Table 8 between NIPPERS, Skip-Graphs, Chord and its newest variations (F-Chord($\alpha$) Stoica et al., 2003, LPRS-Chord (Zhang et al., 2003) BATON (Jagadish et al., 2005) and its newest variation (BATON$^*$ Jagadish et al., 2006). Our contribution provides for exact-match queries, improved search costs from O(log$N$) in

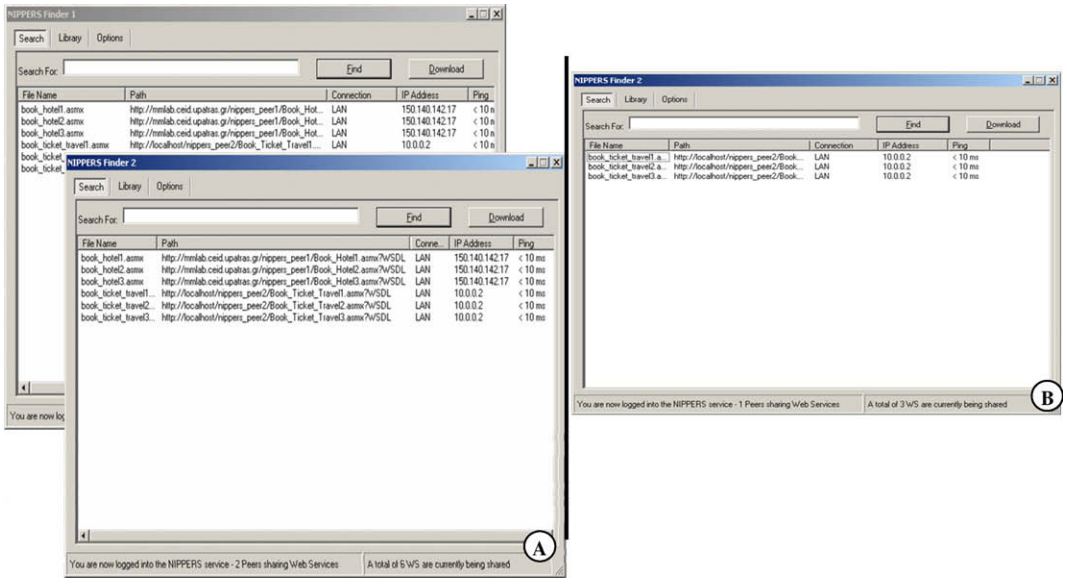


**Fig. 4.** (A) Two NIPPERS clients online listing all available WS. (B) Single only client on NIPPERS network.

**Table 8**
Performance Comparison with Chord, BATON and Skip-Graphs.

| P2P network architectures | Lookup key | Update (insert/delete) key | Data overhead-routing information | Join/depart node |
|---|---|---|---|---|
| CHORD | $O(\log N)$ | $O(\log N)$ | $O(\log N)$ nodes | $O(\log^2 N)$ w.h.p. |
| H-F-Chord ($\alpha$) | $O(\log n/\log\log N)$ | $O(\log n/\log\log N)$ | $O(\log N)$ nodes | $O(\log N)$ |
| LPRS-Chord | Slightly better than $O(\log N)$ | Slightly better than $O(\log N)$ | $O(\log N)$ nodes | $O(\log N)$ |
| NIPPERS | $O(\log\log N)$ expected with high probability | $O(\log\log N)$ expected with high probability | Exponentially decreasing | $O(\log\log N)$ amortized expected with high probability |
| Strong rainbow Skip-Graphs | $O(\log N)$ | $O(\log N)$ | $O(1)$ | $O(\log N)$ amortized |
| BATON | $O(\log N)$ | $O(\log N)$ | Two (2) nodes | $O(\log N)$ |
| BATON* | $O(\log_m N)$ | $O(\log_m N)$ | $m$ nodes | $O(m\log_m N)$ |

DHTs, Skip-Graphs and BATON (BATON* requires $O(\log_m N))$ to $O(\log\log N)$ in NIPPERS and adequate and simple solution to the range query problem. Update Queries such as WS registration and de-registration requests are not performed as frequently as a user login and logout in a typical P2P data delivery network. Web Services are software developed to support business structures and procedures which are expected to stay available in the WS discovery registries more than a P2P user session time span. NIPPERS scales very well in the expected case with high probability. In worst-case a global rebuilding of the whole structure may occur, which is also not typically met in WS registry/catalogue implementation cases. Additionally, a fault tolerance schema is available to support with fidelity an elementary Web Services business solution.

### 7.2. Simulation and experimental results

In this section we evaluate the NIPPERS protocol by the simulator described in Section 6. The simulator (*source code is available in the following URL:* http://www.ionio.gr/~sioutas/New_Software.htm) generates initially $K$ keys drawn by some $\mu(\cdot)$ distribution. After the initialization procedure the simulator orders the keys and chooses as bucket representatives the 1st key, the $\ln(n)$th key, the $(2 * \ln(n))$th key and so on. Obviously it creates $N$ buckets or $N$ Leaf_nodes where $N = K/\ln n$, each of which stores polylog($n$) keys (or $(\log n)^c$ keys, where $c$ is a constant). The simulator executes the next insertions/deletions as a combinatorial game of bins and balls, ensuring the desirable load balance of each bucket. Finally the simulator uses the lookup algorithm in Table 3.

We compare the performance of deterministic only protocols, meaning the performance of NIPPERS simulator with the Chord simulator presented in Clarke et al. (2000) as well as the BATON* simulator presented in Jagadish et al. (2006) (the theoretical solution of Skip-Graphs presented in Goodrich et al. (2006) uses randomized and not deterministic algorithmic techniques). More specifically we evaluate the maximum load per peer (leaf_peer in proposed solution) and the search path length of these three architectures. In order to understand in practice the maximum load per peer and routing performance of these three protocols, we simulated a network with $N = 2^k$ nodes, storing $K = 100 \times 2^k$ keys in all. Node and key population is the same as in previous experimental evaluations for related work (Clarke et al., 2000; Jagadish et al., 2006). We varied parameter $k$ from 3 to 14 and conducted a separate experiment of each value. Each node in an experiment picked

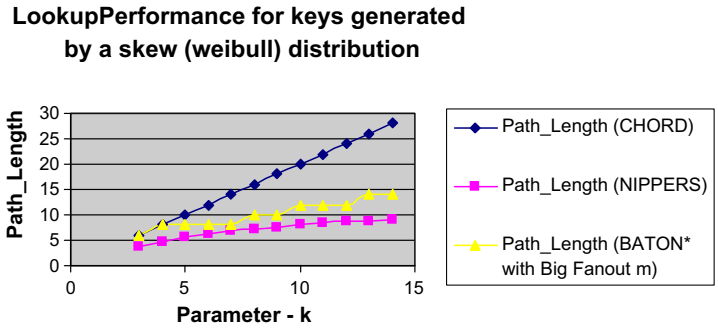


**Fig. 5.** Lookup performance comparison with Chord and BATON* (with Big Fanout) when the inserted/deleted keys draw a skew (weibull) distribution.

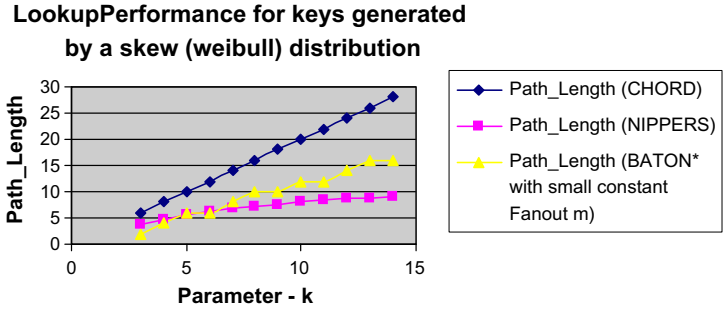


**Fig. 6.** Lookup performance comparison with Chord and BATON* (with small constant Fanout) when the inserted/deleted keys draw a skew (weibull) distribution.

a random set of keys to query from the system, and we measured the path length required to resolve each query. For the experiments we considered synthetic data sets. Their generation was based on three bad distributions the behaviour of which although belongs to smooth family it is also far away from the known good distributions like uniform or regular. It concerns the Weibull, Beta and Normal distribution respectively.

Weibull is a skew density as well as Beta is similar to Gaussian density. For theoretical interest every $(f_1, f_2)$-smooth density can be transformed to real world skew distributions like zipfian, power law or binomial by restricting the mathematical form of $f_2$ function. For more details see the pages 391 and 392 of Kaporis et al. (2006). The figures below present both the path length for lookup queries and the maximum load per peer of each Leaf node when the inserted/deleted keys of the system draw one of the above smooth like functions.

Figs. 7, 10 and 13 show that NIPPERS and BATON* have the most uniformly distributed load per peer, while the parameter – $k$

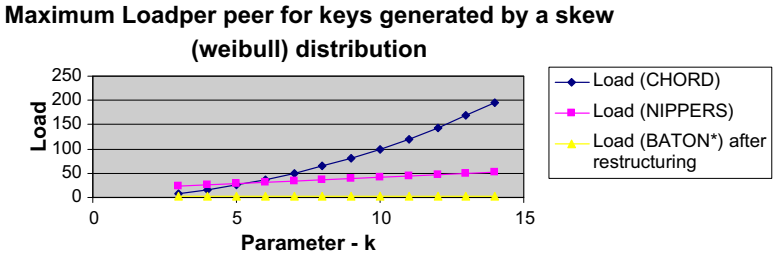


**Fig. 7.** Maximum load per peer comparison with Chord and BATON* when the inserted/deleted keys draw a skew (weibull) distribution.

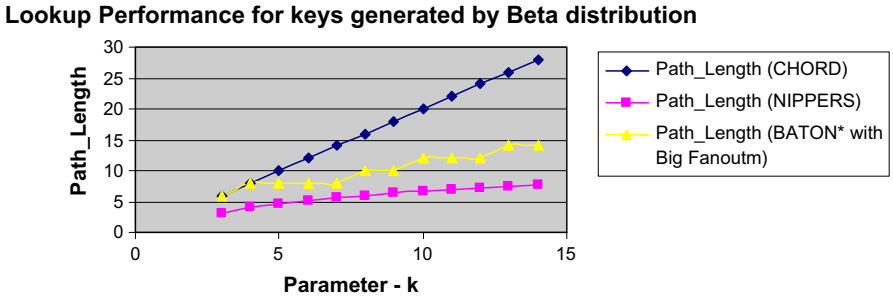


**Fig. 8.** Lookup performance comparison with Chord and BATON* (with Big Fanout) when the inserted/deleted keys draw beta distribution.

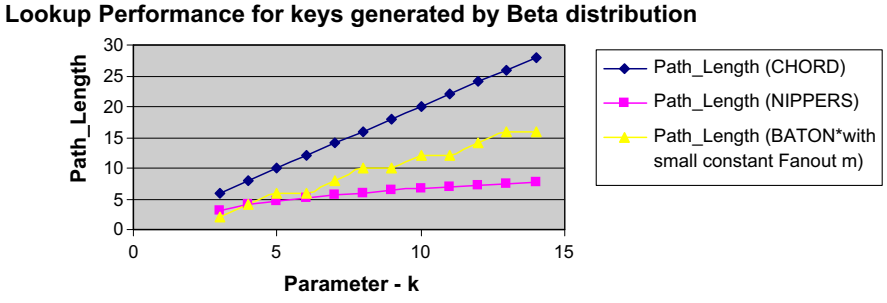


**Fig. 9.** Lookup performance comparison with Chord and BATON* (with small constant Fanout) when the inserted/deleted keys draw beta distribution.

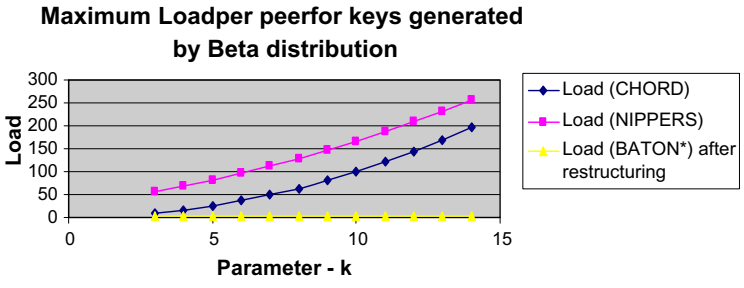


**Fig. 10.** Maximum load per peer comparison with Chord and BATON* when the inserted/deleted keys draw beta distribution.

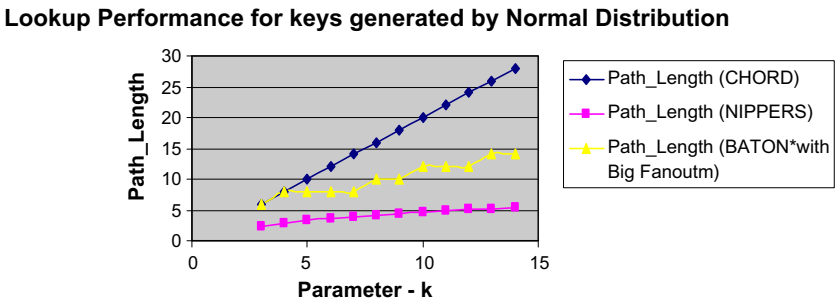


**Fig. 11.** Lookup performance comparison with Chord and BATON* (with Big Fanout) when the inserted/deleted keys draw normal distribution.

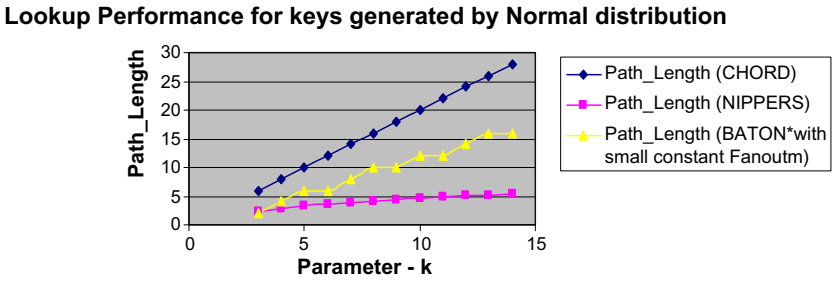


**Fig. 12.** Lookup performance comparison with Chord and BATON* (with small constant Fanout) when the inserted/deleted keys draw normal distribution.

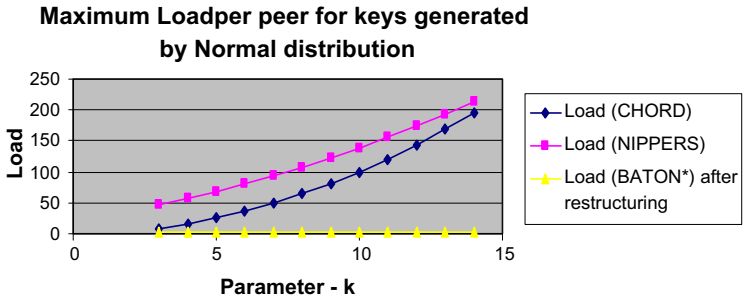


**Fig. 13.** Maximum load per peer comparison with Chord and BATON* when the inserted/deleted keys draw normal distribution.

increases, since Chord's curve is the most acute of the three. Considering now that the depicted load of BATON* system is after restructuring, we conclude that NIPPERS outperforms in overall. In BATON*, if a node is overloaded, it tries to do load balancing with its adjacent nodes first. If there is no lightly loaded adjacent node, it then tries to find a lightly loaded leaf node to do load balancing. Once such a node is found, that node has to perform a forced leave at its current position and a forced join at the new position to share the workload of the overloaded leaf node by leaving the current position and joining in the new position. The tree structure may be imbalanced after that. Thus, network restructuring is triggered if necessary. In the contrary, NIPPERS does not require restructuring for load balancing, not even for the case of joining/departure of nodes, since with high probability rebuilding of the whole structure will never be occurred.

The lookup efficiency of our system is obvious (see Figs. 5, 6, 8, 9, 11 and 12) and in accordance with the aforementioned theoretical complexities. In all experiments we have done NIPPERS outperforms Chord and its variants. When the Fanout m of BATON* is big and specifically a logarithmic function then NIPPERS is a little better than BATON*. When the Fanout m of BATON* is a small constant then our system outperforms BATON* by a wide margin.

**Remark 4.** Although NIPPERS has the most uniformly distributed load per peer while parameter $k$ scales, it appears to have more loaded peers (leaves) than the other solutions. The latter stems from the fact that Nippers stores the real information on leaves, while the other redundant nodes are used for fast routing only. As a consequence, the existence of hot spots is a real fact and this holds not only for the leaves but also for every node of NIPPERS index. To clarify this, just remember the upward and downward walks of NIPPERS lookup algorithm. According to this algorithm, if we are located at leaf_peers belonging to the left subtree $T_l(u)$ of a tree rooted at node $u$ (let say it $T(u)$), and we are looking for keys (services) belonging to the right subtree $T_r(u)$, NIPPERS routes all the searching processes from the same node $u$. This fact reduces the overall fault tolerance and is the main drawback of NIPPERS index.

## 8. Conclusions

The convergence of P2P and Service Oriented Computing has provided exciting new opportunities: WS provide the universal information architecture to solve functional integration problems, and P2P provides the network infrastructure architecture that

enables direct and fault tolerant access to computational resources. One of the most promising potentials of this convergence is the decentralisation of WS discovery which will increase fault tolerance and search efficiency.

A Web Service discovery structure over a P2P network needs to determine the node that stores the Web Service item which satisfies a range criterion. In this paper we introduced and analysed NIPPERS, a protocol that solves this challenging problem in decentralized manner. Current work includes the implementation and experimental evaluation of NIPPERS for large scale WS discovery when the insertion/deletion of WS items draw smooth's family distributions. It offers a powerful primitive: given a key/range of keys, it determines the node/nodes responsible for storing the key's value/values, and does so efficiently. In the steady state, in a $N$-node network, each node maintains routing information for an exponentially decreasing, while we get-down the grid structure, number of other nodes, and resolves all lookups via $O(\log\log n)$ messages to other nodes. Updates require only $O(\log\log n)$ amortized number of messages. The dynamic way (combinatorial game of ball into bins (Kaporis et al., 2003) according to which the NIPPERS was structured ensures that the worst-case re-organization of the whole p2p structure with high probability will never be occurred. This new solution outperforms the most popular infrastructures used directly or as a basis for many solutions for P2P WS discovery including Chord (and some of its successors), BATON (and it's successor) and Skip-Graphs.

Future work includes the construction of a more fault tolerant NIPPERS protocol as well as it's application in other domains such as cooperative file sharing, time-shared available storage systems, distributed indices for document and large-scale distributed computing platforms.

## References


Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R., 2003. P-Grid: a self-organizing structured P2P system. ACM SIGMOD Record 32, 29–33.

Adamopoulou, P., Sakkopoulos, E., Tsakalidis, A.K., Lytras, M.D., 2007. Web service selection based on QoS knowledge management. Journal of UCS 13 (9), 1138–1156.

Andersson, A., Mattson, C., 1993. Dynamic interpolation search in $O(\log\log n)$ time. ICALP'93, 15–27.

Androutsellis-Theotokis, S., Spinellis, D., 2004. A survey of peer-to-peer content distribution technologies. ACM Computing Surveys 36 (4), 335–371.

Ballinger, K., 2003. .NET Web Services: Architecture and Implementation. Addison-Wesley.

Baresi, L., Di Nitto, E., Ghezzi, C., Guinea, S., 2007. A framework for the deployment of adaptable web service compositions. Service Oriented Computing and Applications, vol. 1, No. 1, Springer, pp. 75–91 (April).

Benatallah, B., Hacid, M.S., Leger, A., Rey, C., Toumani, F., 2005. On automating Web services discovery. VLDB Journal 14, 84–96.

Borenstein, J., Fox, J., 2004. Semantic discovery of web services: a step towards fulfilment of the vision. Web Services Journal. <http://www.sys-con.com/webservices>.

Clarke, I., Sandberg, O., Willey, B., Hong, T.W., 2000. Freenet: a distributed anonymous information storage and retrieval system. In: ICSI Workshop on Design Issues in Anonymity and Unobservability, 2000, pp. 46–66.

Cordasco, G., Gargano, L., Hammar, M., Negro, A., Scarano, V., 2004. Non-uniform deterministic routing on F-Chord($\alpha$). In: International Workshop on Hot Topics in Peer-to-Peer Systems, 2004, pp. 16–21.

Crainiceanu, A., Linga, P., Gehrke, J., Shanmugasundaram, J., 2004. P-Tree: A P2P index for resource discovery applications. WWW2004, 390–391.

Diamadopoulou, V., Makris, Ch., Panagis, Y., Sakkopoulos, E., 2008. Techniques to support Web Service selection and consumption with QoS characteristics. Journal of Network and Computer Applications 31 (2), 108–130.

Ganesan, P., Manku, G.S., 2004. Optimal routing in Chord. In: 15th Annual ACM-SIAM Symposium on Discrete algorithms, 2004, pp. 176–185.

Gonnet, G., Rogers, L., George, J., 1980. An algorithmic and complexity analysis of interpolation search. Acta Informatica 13 (1), 39–52.

Goodrich, M.T., Nelson, M.J., Sun, J.Z., 2006. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure, ACM SODA 2006, January 22–26, Miami, FL, pp. 384–393.

Jagadish, H.V., Ooi, Beng Chin, Hieu, Quang, 2005. BATON: a balanced tree structure for P2P networks. VLDB, 661–672.

Jagadish, H.V., Ooi, B.C., Tan, K.L., Vu, Q.H., Zhang, R., 2006. Speeding up search in P2P networks with a multi-way tree structure. ACM SIGMOD, 1–12.

Kaporis, A., Makris, Ch., Sioutas, S., Tsakalidis, A., Tsichlas, K., Zaroliagis, Ch., 2003. Improved bounds for finger search on a RAM, ESA 2003. LNCS 2832, 325–336.

Kaporis, A., Makris, Ch., Sioutas, S., Tsakalidis, A., Tsichlas, K., Zaroliagis, Ch., 2006. Dynamic Interpolation Search Revisited, ICALP 2006, Part I. LNCS 4051, 382–394.

Koloniari, G., Pitoura, E., 2005. Peer-to-peer management of XML data: issues and research challenges. Sigmod Record.

Li, Y., Zou, F., Wu, Z., Ma, F., 2004. PWSD: a scalable web service discovery architecture based on peer-to-peer overlay network. APWeb04 LNCS 3007, 291–300.

Li, F., Yang, F., Shuang, K., Su, S., 2007. Q-Peer: a decentralized QoS registry architecture for web services, in the proc. Service-Oriented Computing – ICSOC 2007. In: Fifth International Conference, Vienna, Austria, September 17–20, 2007, pp. 145–156.

Liben-Nowell, D., Balakrishnan, H., Karger, D., 2002. Analysis of the evolution of peer-to-peer systems. In: 21st Annual Symposium on Principles of Distributed Computing, 2002, pp. 233–242.

Liu, J., Zhuge, H., 2005. A semantic-link-based infrastructure for web service discovery in P2P networks. In: 14th International Conference on World Wide Web, 2005, pp. 940–941.

Makris, Ch., Panagis, Y., Sakkopoulos, E., Tsakalidis, A., 2004. Efficient search algorithm for large scale web service data, RACTI Technical Report No. CTI TR 2004/09/01.

Mayamounkov, P., Mazieres, D., 2002. Kademlia: a peer-to-peer information system based on the xor metric. In: First International Workshop on Peer-to-Peer Systems, 2002, pp. 53–65.

Mehlorn, K., Tsakalidis, A., 1993. Dynamic interpolation search. Journal of the ACM 40 (3), 621–634.

Microsoft .NET Framework. <http://msdn.microsoft.com/netframework/>.

Microsoft Visual Studio .NET. <http://msdn.microsoft.com/vstudio>.

OASIS UDDI. <http://www.uddi.org/>.

Ouzzani, M., Bouguettaya, A., 2004. Efficient access to web services. IEEE Internet Computing (March/April), 34–44.

Overmars, M., Leeuwen, J., 1981. Worst case optimal insertion and deletion methods for decomposable searching problems. Information Processing Letters 12 (4), 168–173.

Panagis, Y., Papakonstantinou, K., Sakkopoulos, E., Tsakalidis, A.K., 2008. Web service workflow selection using system and network QoS constraints. International Journal of Web Engineering and Technology 4 (1), 114–134.

Papazoglou, M.P., Krämer, B.J., Yang, J., 2003. Leveraging web-services and peer-to-peer networks. CAiSE, 485–501.

Ran, S., 2003. A model for web services discovery with QoS. ACM SIGecom Exchanges 4 (1), 1–10.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S., 2001. A scalable content-addressable network. ACM SIGCOMM Computer Communication Review 31 (4), 161–172.

Rowstron, A., Druschel, P., 2001. Pastry: a scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science 2218, 329–350.

Sakkopoulos, E., Makris, Ch., Sioutas, S., Triantafillou, P., Tsakalidis, A., Vassiliadis, B., 2005. NIPPERS: Network of InterPolated PeERS for web service discovery. In: IEEE International Conference on Information Technology: Coding and Computing, 2005, pp. 193–198.

Schmidt, C., Parashar, M., 2004. A peer-to-peer approach to web service discovery. World Wide Web Journal 7 (2), 211–229.

Singh, P.M., Huhns, M.N., 2005. Service Oriented Computing, Semantics, Processes, Agents. Wiley Press.

Sioutas, S., Sakkopoulos, E., Drossos, L., Sirmakessis, S., 2008. Balanced distributed web service lookup system. Journal of Network and Computer Applications 31 (2), 149–162.

Sivashanmugam, K., et al., 2004. Speed-R: semantic P2P environment for diverse Web Service registries. <http://webster.cs.uga.edu/~mulye/SemEnt/Speed-R.html>.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H., 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking (TON) 11 (1), 17–32.

Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N., 2003. Automated discovery, interaction and composition of Semantic Web Services. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 1, 27–46.

Taylor, I.J., Harrison, A., 2005. From P2P to Web Services and Grids: Peers in a Client/server World. Springer.

Tsalgatidou, A., Koutrouli, E., 2005. Interoperability and eServices. LNCS 3543, 50–55.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., Miller, J., 2005. METEOR-S WSDI: a scalable P2P infrastructure of registries for semantic publication and discovery of web services. Information Technology and Management 6 (1), 17–39.

Willard, D.E., 1985. Searching unindexed and nonuniformly generated files in $\log\log N$ time. SIAM Journal of Computing 14 (4), 1013–1029.

Yu, S., Liu, J., Le, J., 2004a. Intelligent web service discovery in large distributed system. In: Yang, Z.R. et al. (Eds.), IDEAL 2004, Lecture Notes in Computer Science, vol. 3177. Springer-Verlag, Berlin Heidelberg, pp. 166–172.

Yu, S., Liu, J., Le, J., 2004b. Decentralized web service organization combining semantic web and peer to peer computing. In: Yang, Z.R. et al. (Eds.), Lecture Notes in Computer Science. Springer Verlag GmbH, pp. 116–127.

Zhang, H., Goel, A., Govindan, R., 2003. Incrementally improving lookup latency in distributed hash table systems. ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems 31 (1), 114–125.

Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D., 2004. Tapestry: a resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22 (1), 41–53.

Zou, Y.F., Ma, F., Li, M., 2004. pXRepository: a peer-to-peer xml repository for web service discovery. Lecture Notes in Computer Science 3251, 137–144.

**Spyros Sioutas** was born in Greece, in 1975. He graduated from the Department of Computer Engineering and Informatics, School of Engineering, University of Patras, in December 1997. He received his Ph.D. degree from the Department of Computer Engineering and Informatics, in 2002. He is now an Assistant Professor in Informatics Department of Ionian University. His research interests include Databases, Computational Geometry, GIS, Data Structures, Advanced Information Systems, P2P Networks and Web Services. He has published over 60 papers in various scientific journals and refereed conferences.

**Evangelos Sakkopoulos** is an Adjunct Assistant Professor at the Computer Engineering and Informatics, University of Patras. He received the Computer Engineering and Informatics diploma degree, the M.Sc. degree in Computer Science and Technology, and the Ph.D. degree from the Department of Computer Engineering and Informatics, University of Patras, Greece, in 2001, 2003, and 2005, respectively. His research interests include Web Engineering, Web Services, e-learning, Web searching, Internet technologies, and mobile Web. He has published more than 60 papers in international journals and conferences in these areas.

**Christos Makris** graduated from the Department of Computer Engineering and Informatics, School of Engineering, University of Patras, in December 1993. He received his Ph.D. degree from the Department of Computer Engineering and Informatics, in 1997. He is now an Assistant Professor in the same Department. His research interests include Data Structures, Web Algorithmics, Computational Geometry, Data Bases and Information Retrieval. He has published over 40 papers in various scientific journals and refereed conferences.

**Bill Vassiliadis** obtained his diploma and Ph.D. degree from the Department of Computer Engineering and Informatics of the University of Patras in 1995 and 2003 respectively. His is currently a post doctoral research fellow at the Hellenic Open University. His research interests include knowledge engineering, knowledge management, intelligent information integration, business modeling and advanced information systems. He has published more than 60 papers in international conferences and journals.

**Athanasios Tsakalidis** is currently a Full Professor and Vice-President of the Department of Computer Engineering and Informatics, University of Patras, Greece. He received the Diploma degree in mathematics from the University of Thessaloniki, Greece, the Diploma degree in computer science, and the Ph.D. degree from the University of Saarland, Saarbuecken, Germany, in 1973, 1981, and 1983, respectively. His research interests include data structures, graph algorithms, computational geometry, expert systems, GIS, medical informatics, databases, multimedia, information retrieval, and bioinformatics.

**Peter Triantafillou** is currently a Professor with the Department of Computer Engineering and Informatics, at the University of Patras, Director of the Network Centric Information Systems Laboratory, and Director of the Software Division. Peter received the Ph.D. degree from the Department of Computer Science at the University of Waterloo, in 1991 and has held professorial positions at the School of Computing Science at Simon Fraser University, and at the Department of Electronic and Computer Engineering at the Technical University of Crete, Greece. Peter's research efforts currently focus on large-scale distributed system infrastructures for content delivery, sharing, and integration, with emphasis on peer-to-peer systems, distributed event-based systems, social networking, and cloud computing. In the recent past he worked on high performance storage systems and on distributed servers, with special emphasis on supporting multimedia applications. Earlier research activities had focused on distributed file systems, multi-databases, and highly-available distributed databases.