**Proceedings of the 17th World Congress**
**The International Federation of Automatic Control**
**Seoul, Korea, July 6-11, 2008**

# A Peer-to-Peer-Based Service Infrastructure for Distributed Power Generation

**Fabian Stäber** [*] **Christoph Gerdes** [*] **Jörg P. Müller** [**]

[*] *Siemens Corporate Technology, Information and Communications, Otto-Hahn-Ring 6, Munich, 81739, Germany (e-mail: fabian.staeber.ext@siemens.com, c.gerdes@siemens.com).*
[**] *Clausthal University of Technology, Julius-Albert-Str. 4, Clausthal-Zellerfeld, 38678, Germany (e-mail: joerg.mueller@tu-clausthal.de)*

**Abstract:** The shift towards decentralization in power generation raises the need for a large-scale control infrastructure to support a high number of distributed power generators. Peer-to-peer computing provides self-organizing, Internet-scale infrastructures being resilient to node failure, which makes peer-to-peer seem a natural basis for a large-scale control application. However, raw peer-to-peer overlays lack certain properties that are crucial in a distributed power generation scenario, like a messaging service, a public key infrastructure for implementing security, etc. Therefore, the peer-to-peer overlay must be complemented by a set of service layer components fulfilling the application requirements.
In this paper, we present an industrial case study where generic service components are applied on top of a peer-to-peer overlay, forming a reliable control infrastructure for distributed power generation. To illustrate our results, we present a detailed evaluation of the level of reliability achieved with the combination of a messaging service, and a replication service.

## 1. INTRODUCTION

Traditionally electricity infrastructures are strictly hierarchical with a top-down flow of electric power from a few large power plants down to a large number of consumers. This situation is changing dramatically. Driven by governmental subsidies small businesses and private households install small and medium power plants (ranging from 1 to 1000kW), generating electric power from solar radiation, wind, waste heat or biogas. This constitutes a paradigm shift in power generation from a central structure towards distributed generation.

The new distributed power generation scenario raises the need for advanced infrastructures in order to maintain a stable balance of generation and consumption. The control infrastructure must be highly scalable and failure resilient, as every single household might potentially have devices for generating power or regulating power consumption in the future.

Peer-to-peer computing provides self-organizing, Internet-scale infrastructures being resilient to node failure. However, the raw peer-to-peer overlay must be complemented with service components, such as components supporting messaging, or components providing a public key infrastructure. In this paper, we present a peer-to-peer based reliable control infrastructure for distributed power generation. We show how the application requirements can be fulfilled composing generic, domain independent service layer components on top of a peer-to-peer overlay.

This paper is organized as follows: In the next section, we describe the overall scenario. Then, we derive the related requirements. Section 4 presents the application architecture, Section 5 gives an overview of related work. Section 6 describes two components, providing messaging and replication. In Section 7, we evaluate the level of reliability achieved with the composition of these two components. Finally, we summarize our results and present an outlook on future work in Section 8.

## 2. USE CASE DESCRIPTION

The control infrastructure is hosted by the utility. The hosts are located at multiple locations and connected to the public Internet. Power generators are connected to the control infrastructure via Internet. The utility uses the control infrastructure to send control commands, and to query status information from the power generators.

From the operator's point of view the control infrastructure provides tree like data structures. The power generators are logically located at the leaf nodes of the tree structures, and the power company uses the trees to address the power generators. The tree structures are a virtual concept and thus independent of physical host computers. Even if underlying hosts fail, new hosts join, or leave the network, the tree structures remain available.

Our infrastructure is capable of providing several data-centric views of the current state of generation and distribution network. Utilities can set the view to include only task specific data. In Figure 1 an example of two parallel tree structures is illustrated.

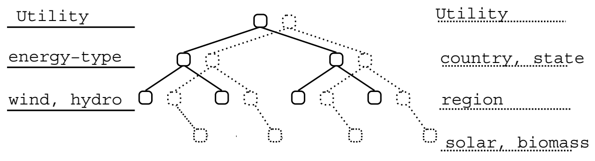


Fig. 1. Energy-Type-Based View of the Power Generators

The tree on the left organizes the generators according to their dependence on weather conditions. The tree on the right is structured by the location of the generators. If the utility needs to multicast a query to all biomass generators, then it is likely to choose the tree organized by energy type whereas if the network operator wants to know the current capacity of all wind power generators in a specific region, then it is likely to use a location-based view to broadcast a query to that region.

## 3. APPLICATION REQUIREMENTS

As motivated in the introduction, we use a peer-to-peer overlay as the underlying infrastructure and set up the tree views on top of that. The key properties of the resulting control infrastructure can be summarized as follows:

(1) *Reliability.* The tree-shaped routing infrastructure must be independent of the underlying hardware, and it must be resilient to failure of hardware. The logical tree view must remain available when hardware fails.
(2) *Scalability.* It is likely that a large share of the households will provide some power generating devices in the future, or that they will at least provide some devices where the power consumption can be controlled to reduce consumption on peak load, e.g. smart meters.
    With the control infrastructure presented in this paper it is possible for the power company to start with a small number of peers, and to increase the number of peers significantly later, while keeping the network traffic load balanced among the peers.
(3) *Messaging, Multicasting, and Aggregation.* The control infrastructure must be able to support multicasting when sending control commands from the power company to the power generators, and it must provide an aggregation tree for receiving periodic reports from the power generators.
(4) *Self-Organization.* The logical tree structure must be independent of the underlying physical infrastructure. It must be possible to add or remove hardware without the need for adapting the tree structure.
(5) *Security.* Unauthorized disclosure, modification, and withholding of information must be prohibited, ensuring confidentiality, integrity, and availability of the system.

## 4. SERVICE COMPONENT ARCHITECTURE

The control infrastructure described in Section 2 is implemented using a service component approach. A combination of generic, domain independent service layer components is applied on top of the peer-to-peer overlay, complementing the overlay with the functionality required by the application.

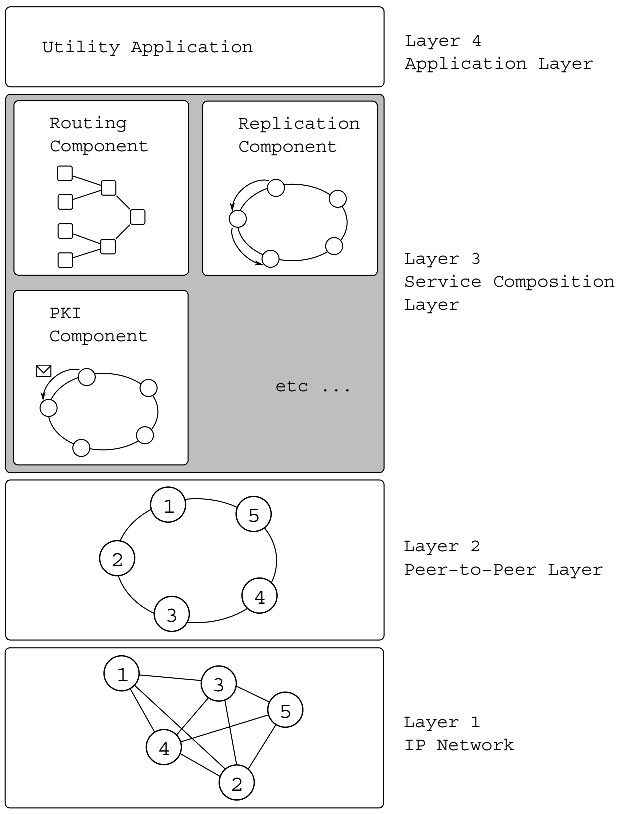


Fig. 2. Layers

The overall architecture is shown in Figure 2. Four layers are defined that build the application:

(1) On the IP layer there are hosts operated by the utility. The hosts are located in different data centers, and are connected through the Internet. In the example in Figure 2, there are five hosts.
(2) On the peer-to-peer layer, each host provides one or more peers in a Distributed Hash Table (DHT). DHTs are structured peer-to-peer systems, where each peer is responsible for a certain range of keywords. When data is stored in a DHT, the responsible peer is found using the keyword associated with that data. The DHT utilized in our approach maps keywords uniformly on the set of peers independent of any network properties.
    The raw peer-to-peer overlay merely provides a routing service, mapping keywords to associated peers. This routing service is resilient to node failure, i.e. if a peer fails, another peer takes over responsibility for the keywords of the failed peer.
(3) The raw routing functionality is used by the service layer components. These are generic, domain independent components being composed to fulfill the application requirements.
    For example, the security requirements are fulfilled by applying a public key infrastructure (PKI) service. Likewise, reliability requirements are fulfilled with a replication service storing backup copies of all data in the DHT. The routing and aggregation service is another very important service in the distributed power generation scenario, as it provides the messaging functionality.

(4) On top of the service layer the actual application is built. From the utility's point of view, a secure and reliable control infrastructure is available, and all details of the underlying DHT are hidden and abstracted by service layer components.

The introduction of a component-based service layer between the peer-to-peer overlay and the application layer allows us to use well-defined components fulfilling application-specific requirements. The component-based approach helps to reduce the complexity of the application.

## 5. RELATED WORK

One of the core tasks of the control infrastructure is to provide a hardware-independent routing and aggregation infrastructure. This infrastructure shares many similarities with distributed XML infrastructures. XPath-like expressions are used to address data. XPath can be used to express singlecasts, multicasts, and broadcasts.

There is a lot of related work addressing the processing of distributed XML on top of peer-to-peer overlays. Most related projects provide some means for processing XPath-like or XQuery-like expressions on the distributed XML. All these solutions share similarities with our work.

The most extensive project addressing distributed XML is *AXML* (Abiteboul et al. [2005a]). The AXML project provides a distributed XML tree, where some nodes are regular *data* nodes, while other nodes are distinguished *function* nodes, representing calls to Web services. AXML supports XQuery processing on the distributed XML structure. If the query processor detects a relevant *function* node, it calls the corresponding Web service and embeds the result in the XML tree. AXML's query evaluation algorithm is described in (Abiteboul et al. [2004]).

Unlike our approach, AXML uses JXTA (Gong [2001]) as the underlying peer-to-peer overlay. There are other projects focusing on DHTs as the peer-to-peer infrastructure. KaDoP (Abiteboul et al. [2005b]) is an implementation of AXML on top of a DHT, with a focus on semantic modeling of the distributed XML data. XP2P (Bonifati et al. [2004]) is a distributed XML store on top of the Chord DHT (Stoica et al. [2001]), providing a distributed XML database.

## 6. QUICK OVERVIEW OF THE MESSAGING AND REPLICATION COMPONENTS

In the distributed power generation scenario, a distributed tree structure is established as a service on top of a peer-to-peer overlay. The tree structure serves as a messaging infrastructure, providing multicasting and aggregation.

In this section, we give a short overview of the implementation of our messaging and replication services. In Section 7, the results are evaluated.

The tree infrastructure is implemented as data being stored in the DHT. Each node of the tree infrastructure is stored as a data resource in the DHT, and each peer in the overlay is able to store several tree nodes. That way, the tree structure may consist of more nodes than the underlying DHT.

A replication service stores backup copies of the data on neighboring peers [1]. If a peer fails, a neighbor can take over the responsibility for the corresponding nodes in the tree structure. That way, a high level of availability can be achieved.

Our routing mechanism uses a *stateless* approach. Each message is tagged with the entire routing information, and each peer can interpret the information and forward the message to the next hop. That way, queries and the corresponding responses are routed independently. If a peer fails after having sent a query, other peers will take over the routing of the response.

Another novelty with our approach is that we use *chunk-based* routing. The tree structure does not need to be traversed hop by hop, but path expressions can be used to jump directly to any position in the tree. Therefore, the inner nodes in the tree are only needed for multicasting and aggregation, but they are not needed when single nodes are addressed directly.

## 7. EVALUATION

As motivated in Section 1, we consider the peers to be potentially unreliable, i.e. peers may fail, or the connection to the Internet may become temporarily unavailable. In this section we evaluate the reliability of the control infrastructure given the potential unreliability of the Internet. The results can be compared to reliability estimates of classical client-server based systems.

We only focus on "clean node failures", i.e. nodes become unavailable or connections become temporarily unusable. This does not include malfunctioning nodes sending invalid messages that could cause invalid routing table entries in the peer-to-peer overlay. Dealing with incorrect messages relates to peer-to-peer security, which is an interesting research topic discussed in (Srivatsa and Liu [2004]). However, it is beyond the scope of this paper.

### *7.1 Churn*

In peer-to-peer terminology, the continuous appearance and disappearance of peers is called *churn*. In this section we evaluate the resilience of the control infrastructure to churn.

Peer-to-peer overlays have originally been developed to support file sharing communities. In typical file sharing scenarios a fraction of the peers joins and leaves the overlay very quickly, which results in comparably high churn rates in these systems (Stutzbach and Rejaie [2006]). As DHT overlays are designed to cope with these high churn rates, we expect the DHT-based control infrastructure to be extremely reliable, as peers in our scenario are run by the power company, and are expected to run in a quite stable environment.

The nodes of the tree structure are published as resources in a DHT. At any given time, there is exactly one peer being responsible for one particular resource, and there are $r$ peers holding replicas of the resource. The responsible peer

[1] In the Chord DHT, neighbor relationships on the overlay are independent of the topology of the underlying hardware.

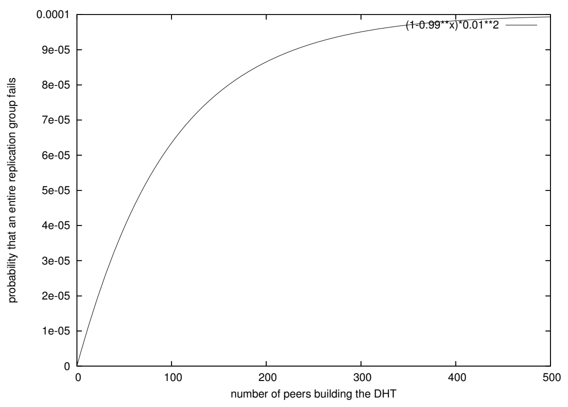


Fig. 3. Probability that an entire replication group fails for $a = 99\%$ and $r = 2$. A deployment with 80 peers would have about half of the maximum failure probability.

plus the peers holding replicas form the *replication group* for that resource. The number of replicas $r$ varies with the DHT implementation, and is typically a value between 2 (as in Rusitschka and Southall [2003]) and 8 (as in Rhea et al. [2004]). Although peers in the replication group are neighbors in the DHT this is generally not true for the physical network where they are uniformly distributed. An important fact to be considered for evaluation.

The failure of an entire replication group results in data loss, i.e. the corresponding node in the tree structure becomes unavailable, and the tree needs to be set up again by the utility. We analyze the probability of data loss in the DHT by deriving a formula for the probability of data loss. We do not use empirical measurements from file sharing networks as done in previous work. The formula enables us to show the difference regarding reliability between classical client-server based systems and peer-to-peer based architectures.

*General Formula.* As a first step, we present a formula that can be applied in both, client-server based scenarios and peer-to-peer architectures. Given a number of $n$ peers (or servers), each of which having an average availability of of $a = 99.999\ldots\%$. Let $r$ be the number of replicas of a resource in the peer-to-peer network (or the number of backup servers in a client-server based scenario), i.e. each resource is stored $r + 1$ times, one original copy plus $r$ replicas. The probability that an entire replication group fails is

$$p = (1 - a^n) \cdot (1 - a)^r$$

The probability has an upper bound of $(1 - a)^r$, and the upper bound is converged with an increasing number of peers $n$. Increasing the number of replicas $r$ decreases the probability of an entire replication group being failed. Figure 3 illustrates the probability for $a = 99\%$ and $r = 2$.

*Interpretation.* In a client-server based scenario, the probability of data loss is usually reduced by increasing the reliability of the server. There are servers with significantly higher availability rates than 99%. Peer-to-peer based solutions are designed for running on cheap, unreliable hardware. These solutions improve the reliability using *self-healing* capabilities. We will now show the impact of self-healing algorithms on the formula above.

All DHTs provide some stabilization protocol including ping messages being continuously exchanged among neighboring peers. A typical time interval between these messages is 10 seconds, as in the Bamboo implementation (Rhea et al. [2004]). If more than $t$ consecutive ping messages remain unresponded, then the corresponding peer is considered to be offline, and the DHT protocol takes another peer into the replication group, and copies the resources to that other peer. A typical value for the threshold $t$ is 3.

In total, as a very pessimistic assumption we can say that it takes less than one minute to detect the node failure and to transfer the replicated resources to another peer. After one minute, the replication group is established again[2]. If necessary, the recovery time can be decreased by increasing the frequency of the ping messages.

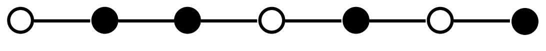
The low recovery time dramatically reduces the probability of data loss in peer-to-peer based systems. In a classical client-server based scenario, a server that fails must be repaired by a service engineer of the service provider hosting that server. Depending on the service level agreement, the provider agrees on how long it takes until the service engineer reacts. This results in a long time interval where one server is missing.

If the guaranteed reaction time of the service engineer is one hour, then the peer-to-peer based system is 60 times faster when replacing failed peers. In the formula above, this means that the probability of data loss is reduced to $p = (1 - a^n) \cdot (1 - a)^r / 60^r$. This explains why peer-to-peer systems are able to provide a high level of reliability without using highly available hardware.

## 7.2 Massive Node Failure

In the section above, we studied the probability of data loss with regular churn, i.e. where the failure of two different peers is independent of each other. This was valid since the utilized DHT maps resources uniformly to peers and hence independent of their physical location in the network. However, there are scenarios where several peers fail at the same time, e.g. when a blackout occurs, and the peers are all affected by the blackout.

In this section we consider the following question: If $k$ out of $n$ peers fail at the same time, how much is the probability of data loss? To the best of our knowledge, this question has not been answered yet in related work about peer-to-peer systems.



The probability $p(n, k)$ is $g(n, k)/m(n, k)$, where $g(n, k)$ is the number of ways to choose $k$ failing peers without data loss, and $m(n, k) = \binom{n}{k}$ is the number of ways to choose $k$ peers out of $n$. In order to construct a recursive formula for $g(n, k)$, we consider all peers in a line. The black peers

[2] Note that this does not mean that the resource is unreachable for one minute. Switching responsibility from one peer to another peer in the replication group is significantly faster than taking a new peer into the replication group.

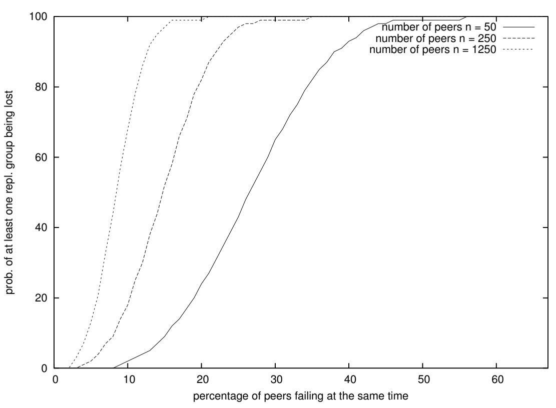


Fig. 4. $k$ of $n$ peers failing at the same time

are the peers that fail, the white peers are the peers that work. The constraint is that there must not be more than two neighboring black peers.

Distributing the failed (black) peers starts at the left. One has three possibilities. First, ○········ the first peer is not failed. In that case, $g(n-1,k)$ possibilities are left. Second, ●—○········ the first peer does fail, but the next one does not. In that case $g(n-2,k-1)$ possibilities are left. Third, ●—●—○········ the first two peers fail, and the third one does not. In that case $g(n-3,k-2)$ possibilities are left. These are all possibilities for the first step.

This consideration leads us to the following recursive formula:

$$g(n,k) = \begin{cases} 0 & \text{if} \quad n < 0 \\ 1 & \text{if} \quad \begin{array}{l} n = k \text{ and} \\ 0 \le n \le 2 \end{array} \\ \begin{array}{l} g(n-1,k) \\ +\, g(n-2,k-1) \\ +\, g(n-3,k-2) \end{array} & \text{otherwise} \end{cases}$$

$n$ decreases with each step of the recursion, and the recursion terminates when $n < 0$. Therefore, it is clear that $g(n,k)$ is well-defined for all $n, k \in \mathbb{Z}$. The formula yields $g(n,k) = 0$ for $n < k$, which means that there is no way to distribute a number of failing peers $k$ that is greater than the overall number of peers $n$.

Figure 4 shows the probability of data loss depending on the percentage of the peers that fail at the same time. The $x$-axis is cut at 66%, because if more than 2/3 of the peers fail the probability for data loss is always 100%.

For small $n$, the formula above produces slightly different results than a simulation. This is because the actual DHT is a ring structure, and not a line. Therefore, $g(n,k)$ counts three cases that would lead to data loss in a ring structure: First, one peer fails at the beginning of the line and two peers fail at the end of the line. Second, two peers fail at the beginning of the line and one peer fails at the end of the line. Third, two peers fail at the beginning of the line and two peers fail at the end of the line. These three cases would lead to data loss if we had a closed ring. However, these three additional cases become insignificant for large $n$.

*Locality.* The consideration above is based on the assumption that the distribution of the replicated resources is independent of the distribution of the peer failures. This is true if the underlying DHT algorithm is based on the Chord protocol (Stoica et al. [2001]), because in these implementations the peers forming a replication group are determined by their peer-id, which is a “random” value.

However, if the underlying DHT algorithm is based on Pastry (Rowstron and Druschel [2001]) or Tapestry (Zhao et al. [2001]), then the replication groups are formed by peers that are close to each other in terms of the underlying network topology. The result of locality-aware replication is that peers being physically located in the same data center are likely to form a replication group. In case of a black out in the data center, the entire replication group is lost.

Therefore we propose to choose Chord as the underlying DHT infrastructure, and to sacrifice the better replication-performance for better reliability in case of blackouts.

### *7.3 Race Conditions*

In the last sections, we focused on the probability that an entire replication group is lost because of peer failures. However, the control infrastructure might be temporarily unstable even if no entire replication group fails. In this section, we will focus on these temporary anomalies.

In case a peer becomes unavailable, queries for that peer are routed to the peer's successor in the replication group. However, it might happen that the original peer becomes available again shortly afterwards. This leads to ambiguous situations when responses are routed back to the root of the tree structure. It might happen that some child nodes report their responses to the peer holding a replicated node, while other child nodes report their responses to the original node.

There are two ways to deal with this situation. First, this situation can be avoided using consensus protocols to elect the responsible peer within the replication group. That way, the peers in the replication group always have a consistent view of the responsibilities. The disadvantage of this approach is that running these protocols requires a significant amount of messages to be exchanged, causing a delay in routing operations.

The second way to deal with this situation is to ignore it. In that case, neither the original peer nor the peer in the replication group will receive responses from all children, so both peers will run into a timeout and forward the incomplete response. As a result, the two incomplete responses will finally reach the utility, and the application at the utility's site can put these responses together to reconstruct the complete response. Considering the low churn rates in the control infrastructure, this should happen very rarely, so this seems to be the best alternative to handle temporary anomalies.

## 8. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a reliable control infrastructure for decentralized power generation. We achieved a high level of reliability without the need of unused stand-by hardware. The network load is distributed among all peers building the routing infrastructure.

We showed that due to the self-healing capabilities of the underlying peer-to-peer infrastructure, the routing infrastructure presented here is more resilient than a classical client-server based approach.

The control infrastructure scales well if there is a huge amount of power generators delivering data, and due to the aggregation of response messages the root node does not become a bottleneck.

Future work includes to investigate the implementation of time constraints, in order to give some guarantees for how long it may take to process a query. Another interesting future topic is further optimization. For example, if a peer is responsible for more than one node in the tree structure, parts of the routing algorithm could be skipped.

Further research must be invested in the aggregation capabilities of the infrastructure. So far, only responses for given queries are aggregated, which means that the routing infrastructure operates in pull mode. It would be interesting to investigate how power generators could actively push data to the power company, e.g. for reporting threshold violations.

## ACKNOWLEDGMENTS

We would like to thank our colleague Dieter Olpp for the valuable discussion of the evaluation.

## REFERENCES

Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, Ioana Manolescu, Tova Milo, and Nicoleta Preda. Lazy query evaluation for active xml. In *SIGMOD '04: Proc. of the 2004 ACM SIGMOD international conference on Management of data*, pages 227–238, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-859-8. doi: http://doi.acm.org/10.1145/1007568.1007596.

Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: an overview. Technical Report 331, Gemo, INRIA-Futurs, 10 2005a.

Serge Abiteboul, Ioana Manolescu, and Nicoleta Preda. Constructing and querying peer-to-peer warehouses of xml resources. In *ICDE '05: Proc. of the 21st International Conference on Data Engineering*, pages 1122–1123, Washington, DC, USA, 2005b. IEEE Computer Society. ISBN 0-7695-2285-8. doi: http://dx.doi.org/10.1109/ICDE.2005.38.

Angela Bonifati, Ugo Matrangolo, Alfredo Cuzzocrea, and Mayank Jain. Xpath lookup queries in p2p networks. In Alberto H. F. Laender, Dongown Lee, and Marc Ronthaler, editors, *WIDM '04: Proc. of the 6th ACM International Workshop on Web Information and Data Management*, pages 48–55. ACM Press, 2004. ISBN 1-58113-978-0. doi: http://doi.acm.org/10.1145/1031453.1031464.

Li Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, May/June 2001.

Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *USENIX '04: Proc. of the USENIX Technical Conference*, pages 127–140, Boston, MA, USA, June 2004.

Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middleware '01, Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, 2001. ISBN 3-540-42800-3. URL `http://www.springer.com/3-540-42800-3`.

Steffen Rusitschka and Alan Southall. The resource management framework: A system for managing metadata in decentralized networks using peer-to-peer technology. In *Agents and Peer-to-Peer Computing*, volume 2530 of *Lecture Notes in Computer Science*, pages 144–149. Springer, 2003. ISBN 978-3-540-40538-2.

Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threats in structured overlay networks: a quantitative analysis. In *ACSAC '04: Proc. of the 20th Annual Computer Security Applications Conference*, pages 252–261. IEEE Press, 2004.

Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, San Diego, CA, USA, 2001. ACM Press. ISBN 1-58113-411-8.

Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proc. of the 6th ACM SIGCOMM on Internet Measurement*, pages 189–202, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-561-4. doi: http://doi.acm.org/10.1145/1177080.1177105.

Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.