A DHT-based Scalable Approach for Device and Service Discovery

Vlado Altmann, Jan Skodzik, Peter Danielis, Johannes Mueller, Frank Golatowski, and Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Email: vlado.altmann@uni-rostock.de

***Abstract*—Web service technology becomes popular in a growing number of applications. The field of application reaches from smart home over factory automation up to smart cities. One of the main aspects of this technology is a dynamic device and service discovery. For this purpose, the WS-Discovery standard is used. WS-Discovery supports two modes: an infrastructure-based and an ad hoc mode. The infrastructure-based mode requires a service proxy, which represents a single point of failure. The ad hoc mode is based on the multicast and static response time values. Therefore, it shows a low scalability. In this work, we present a novel discovery approach, which bases on distributed hash tables (DHTs). DHTs are often utilized in peer-to-peer networks and represent a well-established technology. The search for devices and services is performed using hash values. The benefit of this approach is a high scalability, deterministic behavior, and flow control. The suggested approach can be applied for networks of any size. Since this approach uses unicast messages, it is not limited to local networks allowing the device and service discovery over multiple subnets or even over the Internet.**

Keywords—Web services; Peer to peer computing; Communication networks; Ad hoc networks; Embedded software

I. INTRODUCTION

The Web service technology dissemination takes place in nearly all areas. This reaches from home over factory automation up to smart cities [1]-[3]. The benefits of this technology are standardized, adjustable, and self-describing interfaces. Since Web services use Extensible Markup Language (XML) for data representation, the messages can be interpreted by machines as well as humans. For embedded environments, specific profiles such as Devices Profile for Web Services (DPWS) were developed [4]. DPWS represents a minimum set of features and Web service standards required to control, monitor, discover, and represent any device. For device and service discovery, the WS-Discovery standard is utilized [5]. It allows discovering all or some sets of devices in two different ways. The first kind of service discovery is infrastructure-based mode or managed mode. This mode requires a central instance – a service proxy. All devices register their services at the service proxy. The searching devices send discovery requests directly to the proxy and get a list of available services in the network. The main drawback of this approach is the single point of failure (SPoF) character of the service proxy. If this central node fails no service discovery is possible in the network. To overcome this problem, an ad hoc mode was developed. In this mode, the searching device sends the discovery messages over multicast to all network participants. Although this mode solves the problem of the SPoF of service proxy, scalability problems arise. The searching device does not have knowledge about the size of the network. Thus, an unpredictable number of responses can arrive in a very short amount of time, statically defined by the standard. The searching device can be overwhelmed by responses similar to Denial of Service (DoS) attack. This can result in buffer overflow or packet rejection. The second drawback of this mode is its limitation to local networks since multicast messages are not forwarded by routers.

In this work, we suggest a distributed hash table (DHT)-based service discovery algorithm. The main benefits of this approach are scalability, use of unicast messages, and flow control [6]. As DHT realization, the Kademlia specification is used [7]. The discovery process runs over a unicast by contacting known devices and gathering new contacts from them. In order to find the desired service, hash values are used. With the aid of the hash values, a proximity to the desired service can be calculated. The searching device asks only contacts that provide a better proximity metric. With each step, the searching device at least halves the distance to the desired service. Thus, the search for a specific service is done with a logarithmic complexity. Since the suggested approach is based on unicast messages, the discovery process can run over different subnets and even over the Internet. The searching device can send any number of parallel requests to speed up the search. Thereby, this device has the control over the incoming data traffic and cannot be overwhelmed with responses. By doubling the network size, only one additional step is required to find the service, which results in a very good scalability even for large scale networks.

Briefly summarized the main contributions of this paper are the following:

- An approach for DHT-based device and service discovery,

978-0-7695-5249-1/14 $31.00 © 2014 IEEE
DOI 10.1109/EUC.2014.23

- Automatic bootstrapping into the network,
- A prototype implementation of the suggested approach,
- Measurement results for experimental setup,
- Comparison with the standard WS-Discovery.

The remainder of this paper is structured as follows. Section II gives a short overview about related work in the field of device and service discovery. Section III describes the principles of DHT. The DHT-based approach for device and service discovery is presented in Section IV. The automatic bootstrapping approach is described in Section V. After the implementation details and evaluation of the suggested solution in Section VI, the paper concludes in Section VII.

II. Related Work

Device and service discovery is a basic function for Plug & Play device connectivity. Two types of discovery can be considered: an infrastructure based and an ad hoc discovery. For infrastructure based discovery, some kind of service broker such as Universal Description, Discovery and Integration (UDDI) is required. In [8], a DHT-based UDDI structure is suggested. This structure is resilient to failures due to the distributed nature of the DHT. However, devices are not part of the DHT but only the service proxies. If a local service proxy fails, the devices cannot access the registry. Thus, a local service proxy still remains a SPoF. In [9], the authors suggest a mixed architecture comprising a service proxy and an ad hoc discovery. Thereby, the whole network is divided into several subnets. To find the devices in a subnet, ad hoc discovery is used. To search across the subnets, the proxy is used. The IP address of the proxy is signaled over Dynamic Host Configuration Protocol (DHCP) as a vendor specific option. This solution counters the SPoF character of the service broker. Though, if the proxy is malfunctioning, the search for services in the affected subnet is impossible. Moreover, depending on the size of the subnet, the searching device can be still overloaded with responses from service providers. Furthermore, the scalability with the rising number of subnets is not discussed. In contrast, the suggested DHT-based approach uses a fully decentralized structure. Thereby, all devices are part of the DHT. By means of unicast messages, the searching device can discover devices in other subnets and take the full control over the traffic flow. A hybrid discovery method is proposed in [10]. However, this approach is limited to wireless sensor grids. In contrast, the proposed DHT-based approach is independent from the underlying technology. In [11], the search for devices is implemented as a network scan. Single network addresses, address ranges, or wildcards can be applied for discovery. This discovery approach scales due to the limitation of the address space. However, the response time of the system increases significantly. The address space can be also reduced by applying semantics to the discovery [12]. Nevertheless, the number of responses and the accruing traffic load cannot be predicted. In this work, the proposed scalable DHT-based approach allows deterministic behavior of service discovery with full traffic control across subnets independent of the number of devices.

III. DHT Basics

Peer-to-peer (P2P) networks were originally developed for file exchange. They can be considered as distributed file storage. The main operations are storing, finding and downloading of files. Depending on the method how files are stored and found different kinds of P2P networks can be distinguished. These are unstructured and structured P2P networks. In unstructured networks, any data can be stored on any peer. Contrary, in structured P2P networks, a relationship between data and peer is established. Thereby, the data determines the peer, where it can be stored.

In order to find some data in an unstructured network, a neighbor peers must be asked. They forward the request to their neighbors and so on. Thus, the network must be flooded with requests. The termination criterion of such search prevents that existing data can be not found if the searching peer and the peer with the data are logically far away from each other. To remedy this problem, a central instance for search requests similar to service broker (centralized P2P network) or some super peers with the knowledge of some part of the network can be applied (hybrid P2P network). However, both approaches insert a SPoF into the P2P network [13]. In contrast, the DHT-based P2P networks do not require a central instance, since the structure is given by the DHT. Moreover, the DHT structure enables a logarithmic complexity in the worst case because only relevant peers are requested during the data search.

The relation between peer and data is determined by means of IDs. The ID of the data is calculated as a hash value from some characteristic information, e.g., the file name. For hash value calculation, the message-digest algorithm 5 (MD5) is often used [14]. Peer IDs are calculated from random values. The responsibility of the peer for some data is defined by means of the search tolerance. It determines the required similarity score between peer and data IDs. There are different methods to determine the similarity score. In this work, we use the XOR metric as defined by the Kademlia specification [7]. The similarity (S) can be determined as shown in Formula 1.

$$S = ID_{peer} XOR\ ID_{data} \quad (1)$$

The peer is then responsible for the data if S is less than the search tolerance value. The XOR metric is also called prefix matching, i.e., the peer ID and the data ID are more similar if more leading bits are the same. The whole network can be represented as a tree, a number line, or as a ring as shown in Figure 1. In the following, the term DHT ring will be used.

In order to communicate with other nodes, every peer maintains a routing table. Contact information of other peers such as IP address, hash value, and XOR distance are stored in it. The routing table is structured as a binary tree. Known peers are inserted into buckets and sorted depending on the XOR distance. In every bucket, the same number of peers can be stored. The maximum number of peers in a bucket can be adjusted by the application. With increasing distance from the own hash value, the buckets cover an increasing part of address space. Thus, every peer knows many contacts in a close proximity and only few distant contacts. An exemplary routing table for a four bit address space is depicted in Figure 2. More information about the routing table can be found in [15]. In order to enter

the network and get some contacts for the routing table, a peer performs a bootstrap. For this purpose, one node must be known. This known node is used as the bootstrap node and is requested to enter the network. The bootstrap node includes the new node in its routing table if its hash value is not already known, and responds with some other contacts from the network. Other contacts can be also contacted to fill the routing table. The routing table is maintained over time in order to keep contacts up to date and remove dead nodes.

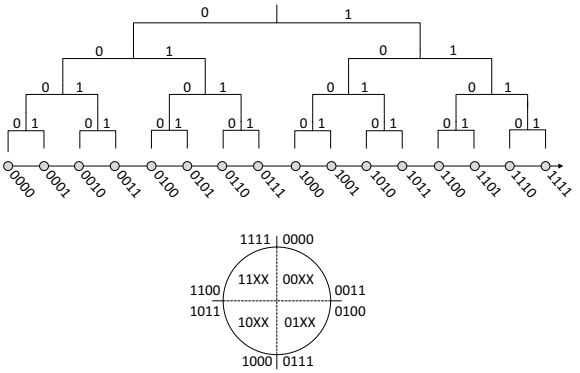

Figure 1: Representation of the structured DHT-based network for 4 bit address space

If a peer searches for specific data, the data ID (hash value) is calculated and some contacts from the own routing table are requested, which are in the search tolerance of the data ID. If such nodes are not available in routing table, the nodes with the highest proximity to the data ID are requested. The requested nodes send new contacts, which yield better proximity to the data hash value. Afterwards, the new contacts are requested. By this means, the searching node reduces the distance to the data ID with every step until the desired node within the search tolerance is found. Moreover, for search speed up, parallel requests can be sent to different nodes.

IV. MULTIKAD

The DHT-based approach was primarily developed for file storage and lookup. The requirements on such system are different from those for service discovery. In order to get the file, *at least one* peer within the search tolerance must be found. If more peers are found, the download can be accelerated. In case of the service discovery, it is important to find *all* service providers in order to select the desired one. Thus, in many automation scenarios, it would not be sufficient to find only some or many devices, e.g., printers but not all. The second issue concerns the hash value assignment. In Kademlia, any data can be theoretically assigned to any peer depending on the search tolerance. In automation scenarios, a device can provide only its own specific service. The third issue concerns hash value calculation. In order to calculate a hash value for the service, some distinguishing attribute such as the service name is required. However, since different devices can provide the same service, they all will map to the same hash value. This would lead to a hash collision. In the original Kademlia specification, collision resolution is not provided.

A. Architecture

As stated above, in order to provide an efficient and scalable solution based on the DHT, three issues need to be addressed, which are not part of it in current solutions:

- Find all service providers,
- Assign hash values to devices,
- Resolve hash collisions.

In order to find all service providers, they must have different hash values because only one node per hash value can be entered into the routing table. Moreover, nodes with the same hash value cannot see each other in the network. However, the searching device knows only the desired service name and thus can generate only one hash value. Moreover, if a new device joins the network, it calculates the node ID from the offered service type. By using the straightforward hash value calculation, the node ID will collide with other service providers, which offer the same service and already entered the network.

In order to solve these problems, a two level DHT architecture called MultiKad was developed in this work. The first level DHT ring is named main ring and the second level – subring or service ring. Every ring has its own DHT. The hash values of the main ring represent the service types. The hash values of the service ring represent service providers. Every device is a member of both rings. Thus, it has two hash values: a main hash value and a service hash value. The main hash value is calculated with MD5 from the service name. The service hash value is calculated from a random value. For every hash value, one routing table is required on the device. Providers of the same service are settled on one hash value on the main ring. The main ring can be seen as a connection structure

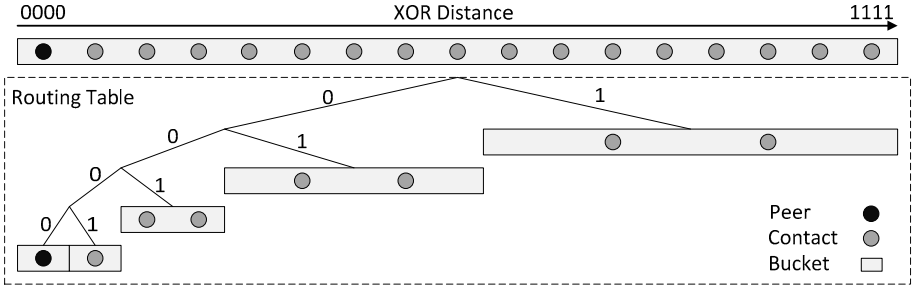

Figure 2: Routing table of a peer for 4 bit address space. Each bucket can contain a maximum number of 2 peers in this example.

between the service rings. The suggested architecture is depicted in Figure 3. The print service is exemplarily emphasized.

This architecture perspective is only seen from the outside. From the device perspective, the service ring is bound to the main ring over one device only because solely one entry in a routing table is possible per hash value. Thus, a device sees other devices of the own service type in the service ring and one device per service type in the main ring (see Figure 4). However, the subring is not bound over one device to the main ring but every node of the subring is bound to different devices of other subrings over the main ring. The DHTs of the both rings are separated from each other. The devices can switch the level by using corresponding hash values. In order to separate the requests from each ring, different ports are used.

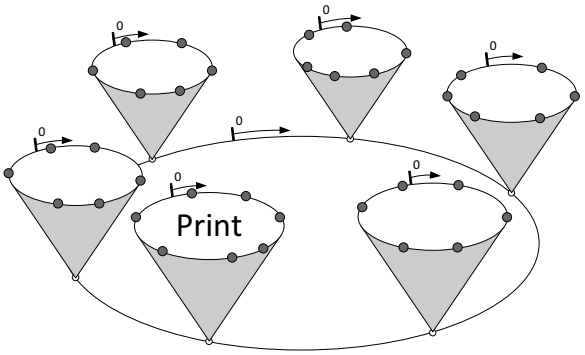

Figure 3: MultiKad architecture

Especially in large scale networks, every device knows some subset of other devices and a complex linkage between subrings via the main ring emerges. Thus, the subrings will be not disconnected from the main ring if devices leave the network. Even in the worst case, if for some reason a subring became separated from the main ring, it will be reconnected again during the maintenance phase. The maintenance phase is part of Kademlia. More information can be obtained from [15].

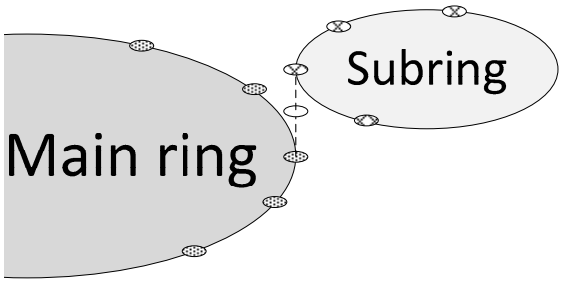

Figure 4: MultiKad architecture from the device point of view

B. Network joining

In order to enter the network the node has to go through the bootstrap process. One contact (bootstrap node) must be known to start bootstrapping. This process requires three steps in MultiKad.

Step 1 is a regular Kademlia bootstrap. The bootstrap node is requested with the main hash value. The main hash value is generated by means of MD5 from the own service type. The response contains several contacts from the main ring. If the hash value of the node is not stored in the routing table of the bootstrap node, it will be added to it. Otherwise, the bootstrap node keeps the old entry and sends its contact information to the joining node. In this case, the step 2 can be skipped.

Step 2 is the search for the own hash value. For this purpose, the contacts from the step 1 are requested. The search tolerance is set 1, i.e., the hash values must match exactly. The search is only successful if one node of the own service type is found. If no other node with the same hash value is found, the bootstrapping is finished. The joining node is the first node in its service type. Otherwise, the joining node continues with step 3.

Step 3 involves a level switch. The node from the step 2 is now the bootstrapping node for the subring. The joining node sends a request to it with a service hash value. The service hash value is generated from a random value. As a response, the joining node gets several contacts from the subring. These contacts are also requested for further new contacts in order to fill the routing table and register own service hash value on other nodes. Therewith, the bootstrapping is done and the joining node is now part of the network. It continues then with the maintenance to keep its routing table up to date.

C. DHT-based discovery

DHT-based discovery is initiated by the client, which wants to find some specific service type. The client sets up a discovery routing table for this purpose and a special client hash value. The client hash value will not be added to the routing tables of other nodes and is not part of the network. Similar to the network joining, the service discovery runs in three steps.

Step 1 starts with one known node from the main ring. The client sends the bootstrap request to this node and receives first contacts from the main ring. Thereby, the client hash value is not added to the routing table of the bootstrap node (see Figure 5, item 1).

Step 2 is the search for the desired service type. The name of the service type is hashed with MD5 and the resulting hash value is the search criterion. The search is performed with a minimum search tolerance, i.e., the hash values must match exactly. The searching node sends request to contacts, which have the highest similarity to the target hash value. In the response message included contacts are requested again, until the target hash value is found (see Figure 5, item 2, 3). This search has a logarithmic complexity as the distance to the target hash value is at least halved per search step. If the target hash value is not found the searched service type is not available in the network. Otherwise, the search process proceeds to step 3.

Step 3 requires the level switch. The found node will be requested on the subring level to send all known contacts from the subring. In opposite to step 2, this request sets the search tolerance to maximum. All new received contacts are requested again. This process continues until no new contacts are received (see Figure 5, item 4-6). These contacts represent the service providers. Finally, the client has the full list of them and can select the desired service provider as well as access the service in a common way.

The search in step 3 has linear complexity because all nodes have to be requested to provide 100 % of discovered devices. However, several nodes can be requested in parallel. The number of parallel requests depends on the client processing power and can be dynamically adjusted to speed up the process. The worst case behavior of suggested approach is described with Formula 2.

$$O(m,n) = log_2(n) + \frac{m}{k} \quad (2)$$

Thereby, n is a total number of devices in the network, m is the number of service providers of desired service type, and k is the number of parallel requests on the subring.

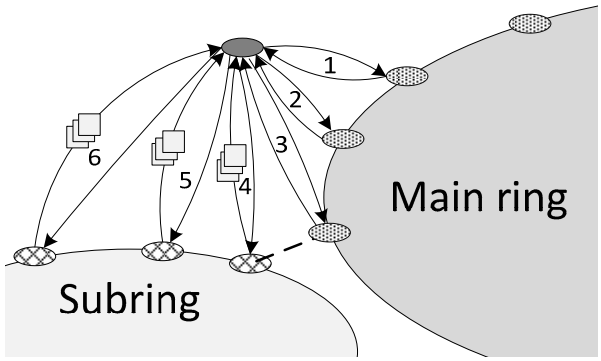

Figure 5: DHT-based discovery

V. AUTOMATIC BOOTSTRAPPING

As described above for network joining and service discovery, one contact must be known. The bootstrapping node serves as an access point to the network. Any node, which is the part of the network, can be used for bootstrapping. Thus, it is sufficient to provide any contact from the network to the joining node or the searching client. Kademlia does not provide mechanisms for automatic bootstrapping. Therefore, the first contact must be manually set up for the new device or searching client. While this is not a problem for a computer, this can be disadvantageous for automation networks, which especially target the Plug & Play functionality. If manual configuration of the bootstrap node is not possible, the bootstrap node can be discovered by means of multicast. As previously mentioned, multicast has several limitations and disadvantages. However, in the suggested approach multicast is only used to find any node and not for service discovery. As multicast is limited to local networks, at least one node, which is the part of the DHT, must be present in the local network.

As the size of the local network is not known, a naive approach to request all devices in the local network to respond, would lead to the described overload of the client. Thus, some mechanism has to be found to limit the number of responses.

In this work, we propose an automatic bootstrapping approach based on incremental search tolerance. For automatic bootstrapping, every node has to generate a bootstrap hash value. This hash value is only used for automatic bootstrapping and not required for manual bootstrapping. The bootstrap hash values are calculated from a random value.

The automatic bootstrap request is sent over the multicast with two significant parameters: target hash value and search tolerance. The target hash value is also calculated from a random value. In the first bootstrap request, the search tolerance is set to minimum. Only one device that has the target value should respond and send its contact for standard bootstrapping. After having sent the request, the bootstrapping node starts the timer. All DHT nodes in the local network receives the bootstrap request and compare the target hash value with the own bootstrap hash value under consideration of the specified search tolerance. If these parameters do not match any node, the bootstrapping request time out will expire. The time out is determined by the bootstrapping node. Afterwards, the search tolerance is doubled and the request is sent again. The doubling of the search tolerance means that one bit less of the target hash value should match the bootstrapping hash value of other nodes. If the second request is not successful, the search tolerance is doubled again and a new request is sent. This procedure is repeated until at least one node responses with its contact. After one or several contacts received, any of them can be used for standard bootstrapping.

In order to test the performance of suggested approach, a test simulation was made. In the simulation, different numbers of hash values were created and the bootstrapping with the incremental search tolerance performed. Thereby, the length of the bootstrapping hash value was changed between 20 and 32 bit. For every parameter set-up, 100 simulation runs was performed. The simulation results are shown in Figure 6 and Figure 7.

It is obvious that the number of responses and the number of leading bits are nearly independent from the length of the bootstrapping hash value. 1.5 responses will be received during the automatic bootstrap request averagely. In the worst case, 11 responses were detected. The simulation results show the scalability of the suggested approach. The searching device cannot be overloaded by the responses. This guarantees a reliable automatic bootstrapping in networks of any size.

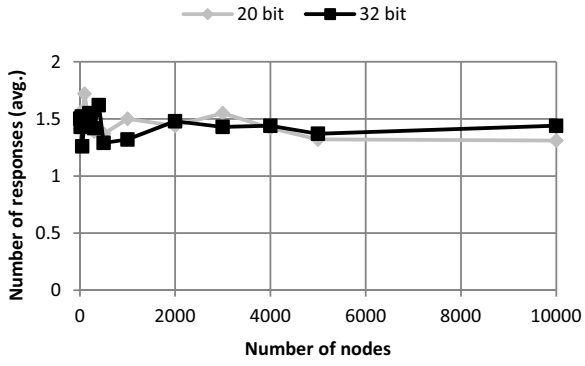

Figure 6: Average number of responses for 20 and 32 bit hash values

The number of leading bits denotes the length of the prefix when the first match occurred (see Figure 7). With the increasing number of nodes, the prefix length increases as well. However, the prefix length is also logarithmically dependent on the number nodes. The length of the hash value determines

the average number of steps required to get the response and thus the average time required for bootstrapping. An example: If the time out for the bootstrap request is set to 50 ms, the bootstrap hash value is 20 bit (about 1 million devices are possible in a local network), and there are 1000 devices in the local network, the new device will need 550 ms to find the bootstrap node. Since automatic bootstrap parameters are set by the client, a shorter timeout can be used to speed up the bootstrapping as well as more bits can be skipped in one step, especially in the first steps. However, the bootstrap process is not a critical operation as it is performed only once for entering the network.

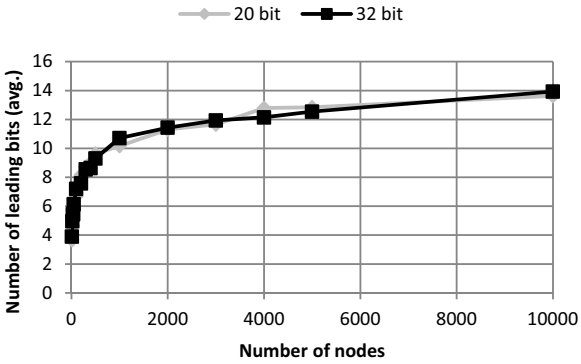

Figure 7: Average number of leading bits for 20 and 32 bit hash values

VI. Implementation and Evaluation

In order to test the suggested DHT-based service discovery approach, a prototype was implemented. As target platform, the evaluation board ZedBoard was used [16]. ZedBoard is based on a Zynq Z-7020 chip, which comprises a Field Programmable Gate Array (FPGA) and an ARM Cortex A9 CPU running at 667 MHz. In the experimental setup, only the ARM CPU is used. The prototype was written in C++ and runs in the real time operating system FreeRTOS for a precise timing measurement. The test implementation has full Kademlia functionality and was extended with the suggested DHT-based service discovery mechanism. Moreover, the standard Kademlia messages were replaced by adapted WS-Discovery messages. For message transport, 1 Gbps Ethernet network is used. A network with 10 devices connected over one switch was built. We measured the average request-response time for two devices during discovery. Thus, an estimated worst case performance for the service discovery for any specific network can be easily calculated with Formula 2. In our test environment, a device requires about 330 µs to request new contacts from the other device. The standard WS-Discovery specification sets aside 750 ms in total for service discovery. In the same time the DHT-based approach would find at least one service in a network with about $1.4*10^{684}$ nodes. If we assume a network with only one service type, 2271 service providers can be discovered in the same time. In more complex networks, the discovery time is composed of service type discovery and the service provider discovery. If we assume a large scale network with 10,000 devices with the remaining communication parameters, the first service provider would be found after approx. 4.6 ms in the worst case. Let us further assume that a device can send 3 parallel requests during discovery. In this case, in the remaining 745.4 ms, more than 6,700 service providers can be discovered. In the real world networks, communication characteristics can highly deviate due to the network load and different device capabilities. However, the searching device can use all available processing power since the response behavior is deterministic. In the standard WS-Discovery, responses from several devices can arrive at the same time. Thereby, the shortage of free resources can occur. Besides the good performance, the main benefit of the suggested approach is that the client can fully control the discovery process avoiding overload and traffic peaks.

VII. Conclusion

In this work, a DHT-based device and service discovery approach was presented. DHT was developed for large scale distributed P2P networks primarily for file storage and download. We used the benefits of DHT such scalability, robustness, absence of single point of failures, and transferred this approach to service discovery. As DHT implementation, Kademlia was used. Since DHTs are based on hash values, the service types are hashed with the MD5 hashing algorithm. The proposed DHT-based service discovery approach uses a two level architecture. The first level represents the main DHT ring. On the main ring, the service types can be discovered. Thereby, one service provider for the desired service can be found. This node is then used as a bootstrap node to the service ring. On the service ring, other service providers can be discovered. The time complexity of the suggested approach is $O(m,n) = log_2(n) + \frac{m}{k}$, whereby n is a total number of devices in the network, m is the number of service providers of desired service type, and k is the number of parallel requests on the subring. For communication between the devices during discovery, only unicast messages are used. Thus, the discovery can be executed over different subnets contrary to the standard WS-Discovery. The searching device has the full control over the discovery process and can adjust the traffic flow corresponding to free resources and processing speed. Consequently, device overload in large scale networks is avoided. In order to apply DHT discovery in Plug & Play networks, an automatic bootstrapping approach is suggested. The approach is based on multicast and an incremental search tolerance. However, the multicast messages are used only for bootstrapping, in order to find one bootstrap node in the local network. The simulation results have shown that 1.5 responses will be received on average independent of the network size. Thus, the bootstrapping device will not be overwhelmed with responses. All proposed mechanisms show good scalability for any network size and can be applied for service discovery in large scale networks.

Acknowledgment

We would like to thank the German Federal Institute for Research on Building, Urban Affairs and Spatial Development (BBSR) within the Federal Office for Building and Regional Planning for their support in this project. This work is partially granted by BBSR within the scope of project “Zukunft Bau”.

REFERENCES

[1] M. Kovatsch, M. Weiss, and D. Guinard, "Embedding internet technology for home automation", *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010.

[2] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, „Web Services in Building Automation: Mapping KNX to oBIX", *5th IEEE International Conference on Industrial Informatics*, 2007.

[3] F. Jammes, B. Bony, P. Nappey, A.W. Colombo, "Technologies for SOA-based distributed large scale process monitoring and control systems", *The 38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, 2012.

[4] D. Driscoll and A. Mensch, "Devices profile for web services version 1.1", OASIS, 2009.

[5] T. Nixon, A. Regnier, V. Modi, and D. Kamp, „ Web Services Dynamic Discovery (WS-Discovery) Version 1.1", OASIS, 2009.

[6] R. Steinmetz and K. Wehrle, „P2P Systems and Applications", *Springer Lecture Notes in Computer Science*, Springer-Verlag, Berlin Heidelberg, 2005.

[7] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", *International workshop on Peer-To-Peer Systems (IPTPS),* 2002.

[8] S. Banerjee, S. Basu, S. Garg, S. Garg, S.-J. Lee, P. Mullan, and P. Sharma, "Scalable grid service discovery based on UDDI", *The 3rd international workshop on Middleware for grid computing*, pp. 1–6, 2005.

[9] S. Poehlsen and C. Werner, "Robust Web Service Discovery in Large Networks", *IEEE International Conference on Services Computing*, vol. 2, pp. 521-524, July 2008.

[10] R. Moreno-Vozmediano, "A hybrid mechanism for resource/service discovery in ad-hoc grids", Elsevier B.V., 2009.

[11] "Communication systems for and remote reading of meters - Part 3: Dedicated application layer", EN 13757-3, 2011.

[12] Zeng ZhiHao; Hu JiPing; Dong Ting; and Wang Yu, "Semantic Web Service Similarity Ranking Proposal Based on Semantic Space Vector Model", *Second International Conference on Intelligent System Design and Engineering Application (ISDEA)*, pp. 917-920, 2012.

[13] J. Eberspächer und R. Schollmeier, „First and Second Generation of Peer-to-Peer Systems," in *Peer-to-Peer Systems and Applications*, Springer-Verlag Berlin Heidelberg, 2005.

[14] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, 1992.

[15] R. Brunner, "A Performance Evaluation of the Kad-Protocol", University of Mannheim, Master Thesis, 2006.

[16] Avnet Inc., "ZedBoard", 2014. [Online]. Available: www.zedboard.org.