Dynamic Search Tolerance at Runtime for Lookup Determinism in the DHT-based P2P Network Kad

Peter Danielis, Jan Skodzik, Vlado Altmann, Lennard Lender, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany, Tel./Fax: +49 381 498-7277 / -1187251
Email: peter.danielis@uni-rostock.de

***Abstract*—For the realization of decentralized communication infrastructures, P2P technology offers an excellent technological foundation to complement or replace existing centralized structures, which take the client-server network model as a basis. In particular, the distributed hash table-based P2P network Kad impresses with high lookup performance, scalability, and resilience against failures and attacks. In addition to these advantages, the Kad protocol needs an extension in terms of its lookup to allow for lookup determinism in an a priori unknown Kad network. Therefore, this paper introduces a dynamic search tolerance, which adapts itself to any network configuration autonomously at runtime. The search tolerance is periodically calculated and all nodes are provided with the updated value. Thereby, the Kad protocol meets the demands for deterministic lookups, which enables dependable and efficient data retrieval, qualifying it as reliable basis of communication infrastructures in, e.g., real-time environments. Results show the high performance of the proposed algorithms for a Kad network with up to 50,000 nodes.**

I. Introduction

Existing communication infrastructures for assuring the operation of Internet services and applications as well as of equipment and networks in automation and home environments are partly characterized by inflexible, centralized, and proprietary structures [1], [2], [3], [4]. Thereby, due to the increasing complexity of the Internet and its ever growing number of users and devices, numerous Internet basic services as well as administration and maintenance services show serious problems concerning scalability and failure resilience. Prospectively, the so-called Internet of Things and Services is even predicted to find its way into the factory [5]. Globally, a networked and real-time capable industrial production is aspired [6]. Thereby, not only the industrial production but also the whole industrial infrastructure shall be intelligently networked. For example, General Electric forecasts for the future that there will be more intelligent devices, which have to be connected to interact with each other dynamically. Most equipment and networks depend on centralized network infrastructures, which take the client-server network model as a basis and do not scale well with future Internet of Things and Services dimensions. Moreover, centralized equipment and networks in home and industrial automation environments, e.g., used for Universal Description, Discovery and Integration (UDDI)-based service discovery show a single point of failure due to their centralized instance [7]. Therefore, they are prone to failures and do not scale with increasing task complexity.

In this regard, P2P technology offers an excellent technological foundation for the realization of efficient decentralized solutions to complement or replace existing structures. Thus, there are already some approaches to develop P2P-based communication infrastructures by utilizing available storage and computing resources [8]. To realize efficient P2P-based communication infrastructures, the choice of a suitable P2P protocol is crucial. By reason of its high lookup performance and minimum maintenance efforts, the Kademlia protocol has been selected as distributed hash table (DHT)-based P2P protocol [9]. Particularly, the Kademlia implementation variant Kad is chosen applied in the eMule client [10].

To qualify the Kad protocol as a reliable replacement of the client-server paradigm in the future Internet of Things and Services, it must meet the demands for *deterministic* lookups, especially in real-time environments [6]. The lookup determinism of Kad depends on a search tolerance (ST), in which a Kad node is considered as responsible node for a given hash value, i.e., for data to be found or stored. In the original Kad implementation, the ST is fixed to a static value at compile time. Such a static ST cannot ensure that there is a responsible node for any data without a priori knowledge of the Kad network and it cannot be adapted to network changes at runtime. Consequently, our contribution was motivated by the idea of introducing a dynamic ST (DST), which is adapted to network changes at runtime so that lookup determinism can be achieved. Briefly summarized, the main contributions of this paper are the following: A conceptual approach as well as simulation and validation results for the determination of a DST to remedy the deficit of a static ST are described.

The remainder of this paper is organized as follows: Section II explains Kad basics. Section III contains a comparison of Kad with state-of-the-art DHT protocols and highlights its benefits. Section IV addresses the concept for the determination of a DST. Section V shows simulation and evaluation results of all algorithmic steps that are necessary for DST determination. The paper concludes in Section VI.

II. Background

For structured data storage, DHTs are applied, which assign a hash value from a common n bit address space to both peers (i.e., nodes) and data. For example, a peer's hash value can be calculated from the peer's IP address; the hash values of data can be calculated from the file name of the data. Depending on their hash values, peers and data are placed in the address space (often represented as ring), which comprises the set of all possible hash values $0 \ldots 2^{n-1}$. Peers are responsible for data, which has a similar hash value like the peers themselves.

978-1-4799-6390-4/15/$31.00 ©2015 IEEE

The function for hash value calculation (e.g., MD4 or MD5) and the bit widths of a hash value are depending on the particular protocol. Also, the protocol defines how to determine the logical distance between two hash values and which similarity in terms of the distance between two hash values is sufficient so that a peer becomes responsible for specific data.

Each peer contains a routing table with contact information on some (not all) other peers. Depending on the hash value of the data to be found or stored, peers with a hash value as similar as possible to the hash value of the data are contacted from the peer's routing table. Thereby, lookup messages usually pass multiple other peers ($O(log_2 N)$ peers in a Kad network with N peers) until they are finally routed to peers, which have a sufficiently similar hash value. For realizing P2P-based communication infrastructures, the DHT-based P2P network Kademlia or accordingly the Kademlia-based implementation variant called Kad offers the best conditions, especially due to its low lookup complexity and maintenance efforts [10], [11].

A. Functionality of Kad

Kad uses an n = 128 bit address space. That is, a 128 bit hash value calculated by the MD4 algorithm in the original implementation is assigned to each peer and each data. The distance between hash values is calculated by the XOR function. For the example of 4 bit hash values, the logical distance of a peer with the hash value $h_1 = 0100_2$ to data with the hash value $d_1 = 0101_2$ equates to $h_1 \ XOR \ d_1 = 0100_2 \ XOR \ 0101_2 = 0001_2 = 1$. In this example, this peer is considered responsible for the data if the ST is set to a value greater or equal to 1.

1) Routing Table: A peer's routing table is organized as binary tree, in which the peer enters contact information of other peers depending on their XOR distances to its own hash value. Thereby, the routing table is arranged in such a way that a peer knows many close but few distant peers in terms of the XOR distance to itself. To join the Kad network, a peer executes a bootstrapping procedure. For this purpose, it contacts a known peer. The contacted peer adds this new peer's contact information into its routing table. Afterwards, the new peer gets to know further peers included in the answer from the known peer. Each peer in the routing table possesses a life time. A maintenance mechanism ensures that peers with expired life time are contacted and removed if they do not reply. Furthermore, some peers are periodically contacted to become acquainted with new contacts (self/random lookup).

2) Lookup: To execute the lookup for target hash values, Kad applies the iterative strategy. A lookup concerns searching for or storing data as well as finding peers. During an iterative lookup, the searching peer (black in Figure 1a) searches for one or more target peers with a hash value sufficiently similar to the target hash value. Sufficiently similar means the peers' hash values are within a predefined ST around the target hash value. All peers along the search path answer the searching peer with new contacts so that the searching peer can communicate with these new contacts itself.

Initially, the searching peer choses peers from its own routing table, which have the smallest XOR distance to the target hash value (contacts from routing table in Figure 1a) but

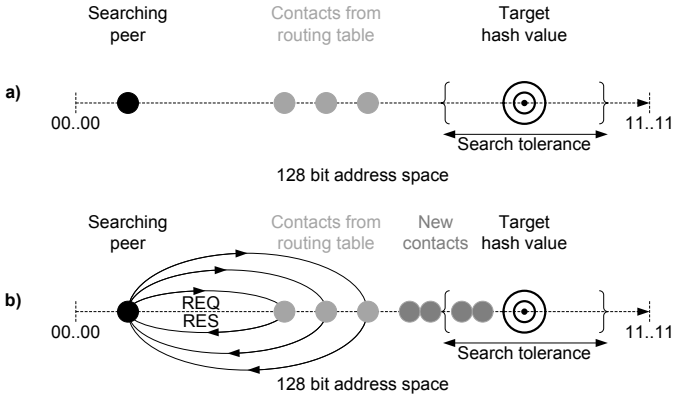

Fig. 1. Kad lookup: a) Initial choice of contacts from the searching peer's routing table. b) New contacts, of which two are within the ST around the target hash value.

do not necessarily need to be within the ST. These contacts are contacted by a request (REQ) and reply with a response (RES). Thereby, the searching peer gets to know new contacts (see Figure 1b), of which two peers are already within the ST.

The peers within the ST are instructed to execute an action like, e.g., storing data. As soon as they have executed the action, a counter is incremented. When this counter reaches a predefined threshold, the lookup procedure is terminated. Alternatively, a timeout cancels the lookup.

B. Lookup Determinism

The lookup determinism in Kad depends on a ST, in which a peer is considered responsible for a target hash value. If the ST was set to a too small value at compile time there would be no responsible peer for the target hash value in Figure 1b. Hence, no lookup determinism would be given, which is inacceptable, e.g., in real-time environments as real-time communication has to be predictable and deterministic. Consequently, the Kad protocol needs an extension in terms of its lookup to ensure its proper functionality.

1) Current ST Definition: The standard value i for the leading bits, which two hash values have to have in common to be considered similar, equals 8 (Kad version 0.49a). Thus, the ST equates to $2^{n-i} = 2^{128-8} = 2^{120}$. This results in a massive data redundancy in case of many peers within this ST as all of them have to store the same data. However, lookup determinism is not necessarily ensured.

2) Static vs. Dynamic ST: In the original implementation of Kad, the ST is statically fixed to an unalterable value at compile time. As Kad was initially used for filesharing, a sufficient number of peers was assumed and high data redundancy was accepted. For the use of Kad for communication infrastructures in an a priori unknown network, this is no longer satisfactory. It is indispensable to dynamically adjust the ST at runtime so that at any time at least one peer is responsible for any data and unnecessary redundancy is avoided. Such a mechanism for automatic ST adaptation is presented in the following sections.

3) Requirements for a DST: However, the dynamic adaptation at runtime requires additional effort. This includes the discovery of all peers, the ST calculation, and its dissemination. Considering data redundancy and communication effort,

which occur in case of a too high ST, and the loss of lookup determinism if the ST is set too small, we argue that the additional effort for a DST is justified. Especially, our approach is suited for hard real-time environments requiring a deterministic network behavior, which cannot be guaranteed by probabilistic gossip-based approaches.

Moreover, it must be taken into account that the ST is not always immediately updated. Joining, leaving, and failing nodes must be detected in adjustable periods of times. If network changes are noticed, the ST has to be recalculated and distributed to all nodes. The longer the periods between two recalculations are set, the higher the probability that the network has changed. The shorter the periods are chosen, the more computation and communication effort is required to determine all nodes, calculate the ST, and disseminate it. However, in a stable Kad network consisting of, e.g., automation devices [12] the period can be set to a few hours without risking a ST being out of date.

As new ST values change node responsibility for data, a node may have stored data, which it is no longer responsible for. Thus, time-to-live values (TTLs) are assigned to data so that their validity automatically expires. The TTL is depending on the scenario and chosen according to data lifetime.

III. State-of-the-Art

There are manifold DHT-based P2P protocols such as, e.g., Chord, Pastry, Kademlia, Symphony, and Viceroy [9]. Each DHT-based protocol provides for a routing table, which contains contacts for lookups. Pastry [13], Kademlia [11], and Symphony [14] contain a flexible routing table. Contrary, Chord [15] and Viceroy [16] use a rigid routing table. Flexibility means that for a given number of nodes, more than one routing table may exist, i.e., there is no rigid and distinct routing table. This flexibility allows for a high lookup performance by using temporary contacts and keeps efforts for the maintenance at a minimum. The lookup performance of Chord, Pastry, and Kademlia amounts to $O(logN)$. In contrast, Symphony and Viceroy have a square-logarithmic complexity $O(log^2N)$. In addition, the search complexity is deterministically determined in Chord, Pastry, and Kademlia; for Symphony and Viceroy, however, it is subject to the probability theory since random processes play a role. In particular, the flexibility of the routing table, high deterministic lookup performance, and good analysis properties are important selection criteria for DHT-based P2P protocols. Since the lookup process with high probability is the most performed operation for all DHT-based networks, the main focus should be on the lookup performance.

Thereby, the lookup complexity is the average number of routing hops that are necessary for a lookup. While in Chord, the complexity firmly depends on the number of nodes N in the network, in Pastry, Kademlia, and Kademlia's implementation variant Kad this complexity depends on a parameter b. The parameter b indicates the number of bits that can be skipped to get closer to the target's hash value per lookup step. That is, the larger this parameter, the closer does one get to the lookup destination per step [9], [17], [11]. Values of the original implementation for b are 4 for Pastry, 5 for Kademlia, and according to the explanations in [17] 6.98 for Kad on average. Furthermore, for the development of Kademlia, a discrepancy of the design of Pastry has been taken up. Pastry's routing metric does not necessarily equal the actual numerical proximity of the node IDs. As a result, two routing phases are required for Pastry, which impairs its lookup performance and results in a complicated formal analysis of its worst-case behavior. Therefore, Kademlia uses the XOR routing metric, which reduces these problems and also allows parallel lookup operations [9]. Kademlia's implementation variant Kad also uses more buckets than originally proposed for Kademlia, which further reduces the average number of necessary hops per lookup [10], [17], [18].

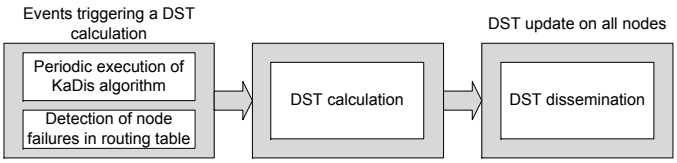

Fig. 2. Procedure of DST determination at runtime.

By reason of its flexibility of the routing table and high lookup performance as well as its analysis properties, the Kademlia protocol has been selected for the realization of efficient decentralized solutions [11], [17]. Kademlia's advantageous analysis properties allow for an easy formal analysis of its worst-case behavior. In addition, as described above it is characterized as advantageous over Pastry regarding the routing and can be optimized by the parameter b in terms of lookup complexity. In particular, the Kademlia implementation variant Kad is chosen, which has been implemented and applied in the eMule client [10].

As mentioned above, the target area for Kad lookups is currently given by a static ST. To *dynamically* adjust the ST, the parameter i has to be adapted at runtime. Contrary to a static ST fixed at design time, it can hereby be guaranteed that for any data, at least one node is responsible. To the best knowledge of the authors, there is currently no mechanism to adapt the ST to changes in the Kad network at runtime.

IV. DST Determination

The ST must be dynamically adjusted at runtime if new nodes join or leave the Kad network or fail. By means of an already developed algorithm (called Kademlia Discovery (KaDis) algorithm [19]), the discovery of all nodes is possible with a high probability without requiring a node to have a priori knowledge of the network. Hence, any node (called master node) can calculate the ST depending on the nodes present in the network (i.e., their hash values) and the width of the address space after having discovered all nodes by means of the KaDis algorithm. To avoid a single point of failure, backup nodes for ST determination should be provided. In addition to the KaDis algorithm, which is executed periodically, the calculating node's routing table is regularly checked for failing nodes and if changes are observed, the ST is recalculated. The initial search tolerance value is set to the width of the address space, i.e., to 2^{128}. After the ST calculation, the calculated value is sent to all nodes. The procedure of DST determination is apparent in Figure 2. Here, the triggers for a calculation, the calculation itself, and the dissemination of the ST are depicted as sequence cascade.

978-1-4799-6390-4/15/$31.00 ©2015 IEEE

A. Triggers for a Calculation

If nodes join, leave, or fail, the ST has to be adapted.

Standard KaDis Algorithm: To be able to do this, a list with all nodes N in the Kad network is provided by the KaDis algorithm periodically with a time complexity of $O(N \cdot log_2 N)$ [19]. If changes have occurred the ST is recalculated.

Between two executions of the KaDis algorithm, the ST may not have the optimal value as changes can have occurred. If a peer joins the network an old ST value can lead to increased data redundancy in the network. This condition is tolerable until the next execution of the KaDis algorithm. However, if nodes fail an old ST value may lead to the loss of lookup determinism and should therefore be recalculated as soon as possible. The routing table of the calculating node is checked for nodes with expired lifetime periodically (every minute in the original Kad implementation of version 0.49a [10]). These nodes are contacted and if they do not reply they are removed from the routing table and from the list with all nodes. Then, the ST is recalculated without execution of the KaDis algorithm. This allows for a faster reaction to node failures. By means of the KaDis algorithm and the detection of node failures, a constant adaptation of the ST at runtime is possible. Thereby, the number of recalculations is kept within a limit as it is solely performed if changes are observed.

Optimized KaDis Algorithm: Due to the parallel nature of Kad, the search for all nodes can be accelerated by utilizing other nodes. This results in a two step procedure consisting of firstly collecting the nodes and secondly returning the information to the originally triggered nodes. Each step is performed in a parallel manner, which results in a performance of $2 * log_2 N$ ($O(log_2 N)$) for both steps. The approach has also been applied to the following dissemination of the DST value.

B. DST Calculation

To sketch the basic idea of the ST calculation, first the standard algorithm is described.

Standard ST Calculation Algorithm: By means of the KaDis algorithm, the hash values of all Kad nodes and their contact data are given. The nodes are sorted by their hash values in rising order with a time complexity of $O(N \cdot log_2 N)$.

The ST has to be calculated in such a way that there are at least $R \geq 1$ responsible nodes for any hash value in the address space. The proposed algorithm for N nodes in an n bit address space starts with the number of leading bits i set to 0 and an initial ST value of 2^n. Iteratively, the whole address space, in which nodes are placed according to their hash values, is divided into 2^i equal segments. Thereby, i is incremented by 1 per iteration. The algorithm terminates as soon as in one of the segments, the desired minimum number of nodes R is no longer present. The ST finally equates to 2^{n-i}, i.e., solely values of powers of two. A total of $u = 2^0 + 2^1 + ... + 2^{i-1} = 2^i - 1$ subdivisions are made as in every step a further subdivision of the obtained segments is carried out. Assuming that the hash values of the nodes are uniformly distributed over the address space, the ST decreases proportionally with increasing network size so that theoretically the following applies: $2^i \sim N$ (with N denoting the number of nodes). Hence, this results in $i \sim log(N)$. By employing this proportionality for the number of performed subdivisions, the following equation is obtained $u = 2^i - 1 \sim 2^{log(N)} - 1 \sim N$. Since the subdivision processes account for the relevant period for calculating proportion of the algorithm, the algorithm's complexity amounts to $O(N)$.

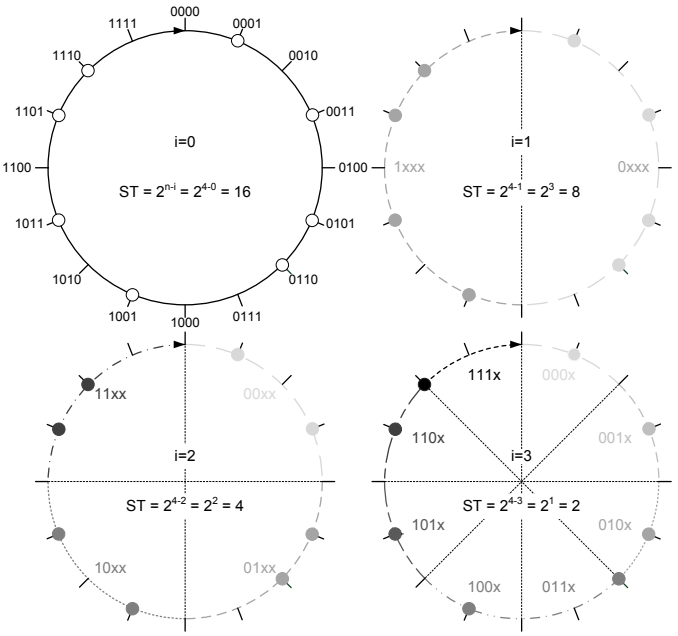

Fig. 3. Iterative ST calculation in a Kad network with N = 8 nodes in a 4 bit address space (n = 4). The whole address space is represented by a ring. Some nodes are exemplarily placed on the ring.

In Figure 3, the algorithm is clarified by means of iteration steps for N = 8 and n = 4. The whole address space is represented by a ring (for i = 0, the hash values are given and after that omitted for clarity). Exemplarily, some nodes are placed on the ring. With each step, the address space is further subdivided and the ST changes its value. In this example, the algorithm terminates as soon as i reaches a value of 3 as there would be less than R = 1 node in any ring segment in the next step. For each ring segment, the leading bits of the node hash values are depicted, which the segment's node hash values have in common. These leading bits are followed by the remaining bits of the hash value represented by x's as the remaining bits are of no interest for the consideration.

Optimized DST Calculation Algorithm: The standard algorithm can be optimized to reduce its execution time significantly. The use of the optimized algorithm does not require a list of ascending hash values so that the time-consuming sorting operation is no longer necessary. The optimized algorithm proceeds according to the following scheme: Before running the KaDis algorithm, a binary tree with a constant depth is created that is based on the minimal computable ST. The deeper the tree is, the smaller the resulting value for the ST can become.

Then, the leading bits of the hash values determined during the execution of the KaDis algorithm are used to mark individual nodes or leaves of the tree. For each hash value, the tree is traversed from the root to one of the leaves. Starting at the root, the most significant bit determines whether to proceed

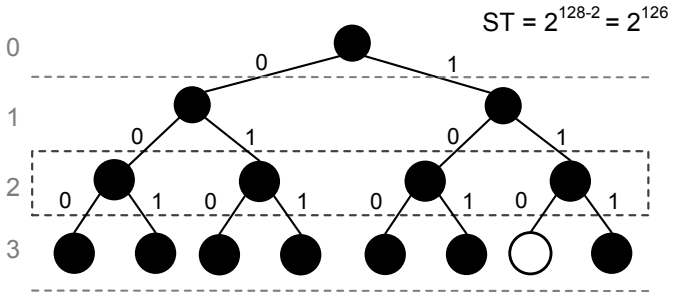

Fig. 4. An example for a binary tree with marked and one non-marked nodes after the execution of the KaDis algorithm.

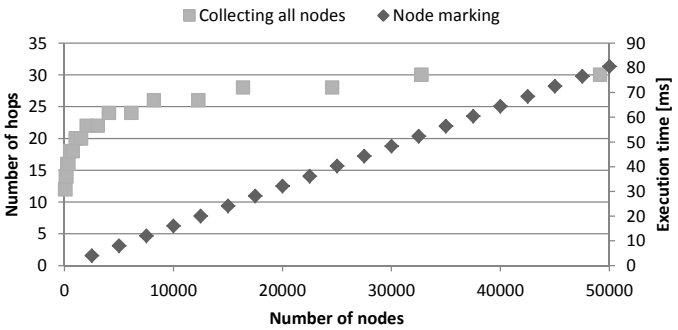

Fig. 5. Time for collecting all nodes within the Kad network and execution time of node marking.

in the left or right subtree. According to this principle, based on the bits of the hash value it is decided at each node whether to continue with the left or the right subtree until a leaf is reached. Thereby, each visited node is marked if this has not already been done. When the operation of marking was carried out with all hash values, the ST can be determined. It has a value of 2^{128-i} for n = 128 whereby $i+1$ is the depth of the tree level, which contains a non-marked node. The minimum ST, which can be calculated using this algorithm, is 2^{128-T} with the depth of the tree T. Only the T most significant bits of the hash values are required and all others can be ignored.

Exemplarily, a binary tree is apparent in Figure 4, in which all available nodes have been marked during the execution of the KaDis algorithm. The tree has a depth of T = 3 so that a ST of $2^{128-3} = 2^{125}$ can be reached. As apparent, the last tree level, which solely contains marked nodes, is level 2. Thus, the ST equates to ST = $2^{128-2} = 2^{126}$ in this example.

The step of marking nodes is already executed during the KaDis algorithm. The time for ST determination in the tree with marked nodes increases with a rising number of nodes N in the network as more levels in the tree can be fully marked and therefore need to be checked. For a ST of 2^{128-i}, i tree levels are fully marked and consequently all nodes had to be checked. Because on tree level t, 2^t nodes are present, at least $\sum_{t=0}^{i} 2^t = 2^{i+1} - 1$ nodes have to be checked. As i logarithmically increases with the number of nodes N, i.e., $i \sim log(N)$, this results in a complexity of $O(2^{i+1} - 1) = O(2^{log(N)+1} - 1) = O(N)$. Hence, the execution time of the algorithm increases linearly with a rising number of nodes in the network.

C. DST Dissemination

After a calculation of a new ST value, which differs from the previous one, each node has to be provided with it.

Standard DST Dissemination: As a list with contact information on all N nodes in the Kad network is already available, they can be contacted directly. Hence, the determined ST can be disseminated with a time complexity of $O(N)$.

Optimized DST Dissemination: This time complexity can even be reduced to $O(log_2 N)$ by determining nodes assisting in the ST dissemination (as done for the time synchronization in [12]). Thereby, all nodes receive the ST value shortly after its calculation. In the period of time between finished calculation and completed dissemination, some nodes use an old ST. However, this period of time is considered negligible.

V. Simulation and Evaluation Results

The algorithms for the DST determination have been simulated for up to N = 50,000 nodes in an address space of n = 128 for the optimized versions of the algorithms. The determined ST varies between a value of 2^{123} for 64 nodes and 2^{115} for 50,000 nodes in the simulations. The ST values have been determined correctly in all cases, which proves that the ST calculation works properly.

A. Execution Time for Optimized KaDis Algorithm

The master node needs to execute the KaDis algorithm to gain knowledge about all other nodes to determine the ST. The optimized KaDis algorithm has a significantly reduced complexity compared to the standard algorithm by using helping nodes to find all other nodes. Therefore, we introduce a collecting factor. For example, a collecting factor of 2 means that each node forwards the search for new nodes to two other nodes. As depicted in Figure 5, with 50,000 nodes we are still below 30 hops, which is about 7.5 ms if we assume that the duration of a hop, i.e., the transmission of a message in the Kad network and the processing time on a node amounts to a total of 250 μs. This value has been derived from an experimental setup comprising prototypes running on ZedBoards, which are connected by a 1 Gbit/s Ethernet switch [20], [12]. The number of hops needed to collect all nodes and to send the information back to the origin node is depicted together with the time needed for marking the nodes. As mentioned above, the node marking is already carried out during the execution of the KaDis algorithm, which results in additional 80 ms for 50,000 nodes. In summary, the overall time amounts to approximately 87.5 ms.

B. Execution Time for Optimized DST Calculation

First, the presented algorithms were implemented for the simulation on a PC. Thereby, they could be easily tested and their complexity could be checked well. After the successful simulation, the time for calculating the DST on a hardware platform suitable for real-time automation environments was determined. A ZedBoard has been used for executing the algorithm [20]. The operating system FreeRTOS (Free Real Time Operation System) was used, which is suitable due to its real-time capability [21].

In Figure 6, the execution time for the optimized algorithm is depicted. After having marked all nodes during the KaDis

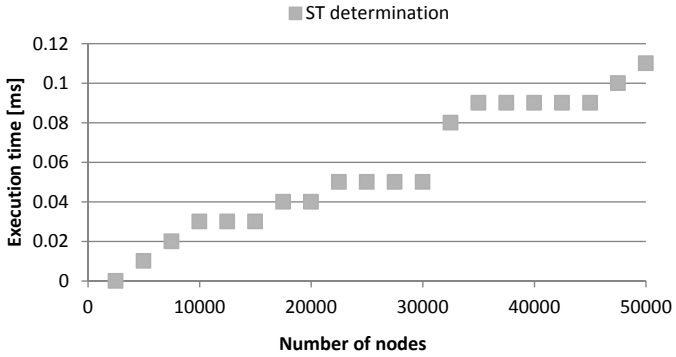

Fig. 6. Execution time of optimized DST calculation consisting of DST determination in the tree with marked nodes.

algorithm, the ST determination solely takes approximately 110 μs for 50,000 nodes.

C. Execution Time for Optimized DST Dissemination

The determined new DST value must be made known to all nodes in the network. To achieve this, the master node contacts other nodes. Thereby, some nodes help to disseminate the DST. The dissemination factor is similar to the previously introduces collecting factor. Thus, a dissemination factor of 2 means that each node forwards the new DST value to two other nodes, of which each sends it to two further nodes until all nodes have been contacted. Factor 2 is sufficient to accelerate the dissemination of the ST compared with the use of a factor of 1. Again, the duration of hops is assumed to be 250 μs, which results in an overall execution time of the algorithm of about 3.75 ms for 50,000 nodes.

D. Summary

The aggregated time values for all steps of the DST determination of approximately 91.36 ms, i.e., below 100 ms for 50,000 nodes justify the practical applicability. Therefore, the algorithms for ST determination can be integrated in the Kad-based network services proposed in [22], [19], [23], [12].

VI. Conclusion

The conclusion of the paper consists in the finding that a dynamic search tolerance for the Kad network at runtime is possible and useful. It allows for lookup determinism, which is of crucial importance in, e.g., real-time environments, and qualifies the Kad network for realizing decentralized communication infrastructures. Thereby, existing centralized structures, which take the client-server paradigm as a basis, can be replaced or complemented. The algorithms have been simulated and successfully validated.

Prospectively, further investigations on how to deal with node failures could be carried out to save the efforts for the execution of the KaDis algorithm in these cases.

Acknowledgement

This work is partly granted by the Research Fund Mecklenburg-West Pomerania, Germany, as well as the European Social Fund.

References

[1] F. Klasen, V. Oestreich, and M. Volz, *Industrial Communication with Fieldbus and Ethernet*. VDE Verlag GmbH, 2011.

[2] M. Felser, "Real time ethernet: Standardization and implementations," in *IEEE International Symposium on Industrial Electronics*, 2010.

[3] EtherCAT Technology Group, "Ethercat technical introduction and overview," 2013. [Online]. Available: http://www.ethercat.org/en/technology.html

[4] M. Felser, "Real-time ethernet - industry prospective," 2013. [Online]. Available: http://www.control.aau.dk/ ppm/P7/distsys/01435742.pdf

[5] Communication Promoters Group of the Industry-Science Research Alliance, acatech - National Academy of Science and Engineering, "Recommendations for implementing the strategic initiative INDUSTRIE 4.0," Industrie 4.0 Working Group, Tech. Rep., April 2013. [Online]. Available: http://www.plattform-i40.de/finalreport2013

[6] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," General Electric, Tech. Rep., November 2012.

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

[8] P. Danielis and D. Timmermann, "Use of Peer-To-Peer Technology in Internet Access Networks and its Impacts," in *IPDPS 2010 PhD Forum, pp. 1–3*, 2010.

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

[10] R. Brunner, "A Performance Evaluation of the Kad-Protocol," Master's thesis, University of Mannheim, Germany, Nov. 2006.

[11] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *IPTPS*, 2002.

[12] J. Skodzik, P. Danielis, V. Altmann, and D. Timmermann, "Time Synchronization in the DHT-based P2P Network Kad for Real-Time Automation Scenarios," in *2nd IEEE WoWMoM Workshop on the Internet of Things: Smart Objects and Services (IoT-SoS)*, 2013.

[13] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM International Conference on Distributed Systems Platforms*, 2001, pp. 329–350.

[14] G. S. Manku, M. Bawa, P. Raghavan *et al.*, "Symphony: Distributed hashing in a small world." in *USENIX Symposium on Internet Technologies and Systems*, 2003, p. 10.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[16] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. ACM, 2002, pp. 183–192.

[17] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," in *INFOCOM*, 2006, pp. 1–12.

[18] S. Khan, A. Gani, and M. Sreekandath, "The routing performance of logarithmic-hop structured p2p overlay," in *Open Systems (ICOS), 2011 IEEE Conference on*, Sept 2011, pp. 202–207.

[19] J. Skodzik, P. Danielis, V. Altmann, J. Rohrbeck, D. Timmermann, T. Bahls, and D. Duchow, "DuDE: A Distributed Computing System using a Decentralized P2P Environment," in *36th IEEE LCN*, 2011, pp. 1060–1067.

[20] Avnet. [Online]. Available: http://www.zedboard.org/

[21] Real Time Engineers Ltd. [Online]. Available: http://www.freertos.org/

[22] P. Danielis, M. Gotzmann, D. Timmermann, T. Bahls, and D. Duchow, "A Peer-To-Peer-based Storage Platform for Storing Session Data in Internet Access Networks," in *World Telecommunications Congress (WTC 2010), pp. 5–10*, 2010.

[23] P. Danielis, J. Skodzik, C. Lorenz, D. Timmermann, T. Bahls, and D. Duchow, "P-DONAS: A Prototype for a P2P-based Domain Name System in Access Networks," in *Design, Automation and Test in Europe Conference and Exhibition (DATE 2010), University Booth*, 2010.