n-Dimensional Border Growth

Daniel Berg, Herwig Unger
Department of Communication Networks,
FernUniversität Hagen, Germany
{daniel.berg|herwig.unger}@fernuni-hagen.de

Abstract: Peer-To-Peer (P2P) networks become more and more present in the consumer area as well as in industrial applications. Especially in the industrial- and the business area, reliable and scalable protocols are needed, that produce low network-overhead and react quickly on any network-changes. In this paper a generalization of the Border-Growth-algorithm is introduced, that improves the network's scalability, its connectivity, and decreases its diameter by providing multiple dimensions, rather than just two of them.

Keywords: P2P, Grid, Scalability, Decentralized Algorithms, Self-Organization

1 Introduction

Decentralized overlay topologies provide characteristics that cannot be achieved with classical client/server architectures. Due to their completely decentralized architectures they can provide wide scalability and fault-tolerance [LCP04, SUL04]. However, maintaining those structures is not trivial. When participants join or leave the network, it has to be ensured that the structure is kept consistent after those operations. This can become a complex task, depending on the complexity of the network's structure [RFH01, CHORD01, STRUCT02].

In [STRUCT02] the Border-Growth-Algorithm was introduced. This algorithm builds a complete, contradiction- and hole-free two-dimensional lattice. Regular lattice-like overlays are very useful for xy-routing based algorithms [RFID07, THERM09, 3DIOS09]. Compared to existing algorithms for overlay structures, like CAN, Chord, Koorde, Pastry or Tapestry the Border-Growth-Algorithm can manage Join- and Leave operations very quickly. As long as the lattice is not broken into two distinct clusters, there are always multiple available paths from any node to another.

This paper aims to improve the scalability of this algorithm by increasing its dimensions from two to n dimensions. It will be showed that this can be achieved without increasing the complexity of the growth-algorithm itself. (The complexity of updating a new node's neighborhood increases linearly with the dimension n.)

Section 2 provides the requirements and a brief description for the 2D-Border-Growth algorithm. In section 3 follows a detailed discussion for the new, n-dimensional version, starting with the motivation in section 3.1 and the modifications made to the new version

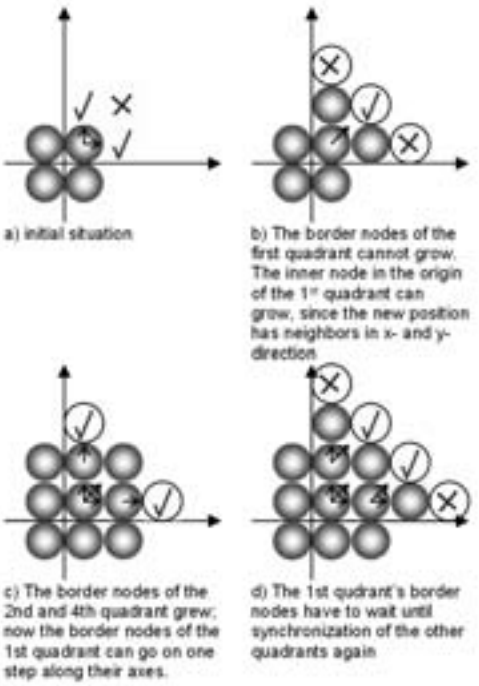

Figure 1: The rules defined by the two-dimensional Border-Growth-algorithm

in section 3.2 Section 3.3 gives the mathematical description, and section 3.4 gives some information about the simulation environment that was used to run the algorithm. Section 4 discusses the simulation results and compares them to the results of the two-dimensional version. Section 5 finally gives a conclusion and a outlook for further work based in the new algorithm.

2 The Border-Growth-Algorithm

2.1 Two-Dimensional Border-Growth

This section gives a brief description for the two-dimensional border-growth-algorithm. The algorithm defines three node types: The **Root Nodes**, that reside at a discrete, virtual coordinate-system's origin in all quadrants, the **Border Nodes** that are positioned along an axis of the virtual coordinate system, and finally the **Inner Nodes** that are all nodes, which are neither Border Nodes nor Root Nodes. The Root Nodes are special cases of the Border Nodes.

Nodes in the two-dimensional lattice have knowledge about their direct neighbors in north-

, west-, south-, and east-direction. To simplify the algorithm, and to increase connectivity, all nodes additionally posses links to their direct neighbors in the diagonal directions northwest, southwest, southeast, and northeast.

The initial state of the structure is illustrated in Fig. 1a): Each quadrant of a virtual, discrete coordinate system has one Root Node in its origin. A Root- or a Border Node is allowed to grow along its axis, if it has a neighbor at the other side of the corresponding axis. This condition is true for the first quadrant's Root Nodes in Fig. 1a and for the first quadrant's Border Nodes in Fig. 1c.

An Inner Node, a Root Node, or a Border Node can grow away from the coordinate system's origin, if the position to which it wants to grow already has neighbors into the x- and y-direction. This condition is fulfilled by the Root Node in Fig. 1b and for two of the first quadrant's Border Nodes in Fig. 1d.

When a new node is accepted as the neighbor of a node in the structure, the new node's neighborhood has to be updated. That means that the surrounding nodes need to update their neighbor-links to accept the new node.

For a detailed mathematical description of this algorithm, refer [BORDER09] and section 3.3 of this paper, which gives a detailed description of the n-dimensional Border-Growth algorithm. Though some modifications had to be made to the new version (see section 3.2), the two-dimensional case is just a special case of the n-dimensional structure.

3 n-Dimensional Border-Growth

3.1 Motivation

Though it is easy to replace a failed node within the structure by another (new or existing) node, such a structure is only able to grow at its borders, which is one reason for why join-operations are very simple and quick. However, growth-scalability in terms of the ratio of the number of all nodes and the number of nodes that can grow, decreases with growing structure size.

In order to lower this effect, the algorithm was modified in that way, that it grows into an arbitrary number of dimensions. Increasing the dimensions of the lattice leads to more advantages: The connectivity of the structure increases, any (inner) node is connected to 2^n other nodes, rather than just to $2^2 = 4$ nodes. Therefore there are more possible paths between any two nodes. Another effect implied by the former one is the reduction of the structure's diameter; the paths between any two nodes of the structure becomes shorter in average.

While the connectivity (which corresponds to the maximum number of neighbors) - and therefore the complexity for updating the neighborhood of a new node - increases exponentially, the complexity of finding a potential position for a neighbor stays constant, independently of the number of dimensions. Even in a n-dimensional grid with $n > 2$, a node just has to consider two dimensions to make a decision, if it can grow or not. The two

dimensions can be selected randomly. A single node gets many more potential positions into which it can grow. If a node recognizes that it cannot grow considering the two chosen dimensions, it can just choose other dimensions and check if the growth-rules are fulfilled for these dimensions.

3.2 Modifications

Some modifications were to be made to the n-dimensional version of the Border-Growth-Algorithm. The first modification is the elimination of diagonal links. Since the number of those diagonal links grows exponentially with the number of dimensions it would not be useful to maintain them within a higher dimensional grid. In 5 dimensions there would be $2^5 = 32$ additional links. The main purpose for doing so was to increase connectivity. But now, better connectivity is achieved by the higher number of dimensions. The mathematical model was relaunched and generalized and now works without diagonal links at all.

To simplify the mathematical description of the algorithm, only the positive quadrants are considered. This makes it necessary to adapt the Border Node growth rule. In the new version a Border Node does not synchronize with a neighbor-node of the neighbor-quadrant, but with its neighbor on the other side within the quadrant. The potential growth-rate of all nodes is equal, and the growth-behavior it symmetric to the origin, so these modification will not significantly change the structure's qualitative growth behavior.

3.3 The n-Dimensional Border-Growth Algorithm

This section gives a detailed mathematical description of the new n-dimensional border-growth-algorithm. Let $n \in N, n > 1$ be the grid's dimension and $t \in N_0$ the discrete time needed, to clearly identify multiple join-operations that occur at the same time. To refer to direct and indirect neighbors of a node, linear combinations of normalized unit vectors are used: $J_+ = \{\vec{j}_1, .., \vec{j}_n |\ \vec{j}_i = (x_1, .., x_n), x_k = \begin{cases} 1, if\ k = i \\ 0, if\ k \neq i \end{cases}\}$ is the set of unit vectors along the positive main axes and $J_- = \{-\vec{j}_i |\ \forall 1 \leq i \leq n\}$ the set of unit vectors along the negative main axes. $J = J_+ \cup J_-$ is the set of unit vectors along all main axes.

The network itself can be described as a graph G with a vertex-set V, and an edge-set E. Since the graph describes a network that changes over time, V and E, and therefore G depend on the discrete time t: $G(t) = (V(t), E(t)), E(t) \subset V^2(t)$. Every vertex of the graph describes a network-node. The function pos associates a n-dimensional position to every node $v \in V(t)$: $pos |\ V(t) \rightarrow \mathbb{N}_0^n,\ \ pos(v_i) = (x_1, ..x_n)$. The function N associates a node $v \in V(t)$ with a neighbor $N(v, \vec{j}_i, t)$ for each $\vec{j}_i \in J$ at time t. If v does not have a neighbor in $\vec{j}_i$, then $N(v, \vec{j}_i, t)$ is 0: $N |\ (V(t), J, t) \rightarrow V(t) \cup \{0\},\ \ N(v, \vec{j}_i, t) = \begin{cases} \text{neighbor of } v \text{ in direction } \vec{j}_i \\ 0,\ if\ \ \nexists w \in V(t) \text{with } pos(v) + \vec{j}_i = pos(w) \end{cases}$

The growth-process starts at $t = 0$. The only node at start-up is a Root Node v_0 residing at the coordinate-system's origin. Since v_0 is the only node, no neighbor-relationships are established yet: $t := 0,\ V(t) := v_0,\ E(t) := \{\},\ pos(v_0) = (0, .., 0)$. We classify the nodes that joined the structure into three types. The **Root Nodes** that reside at the center of the virtual, discrete coordinate system: $V_0(t) = \{v_0|\ v_0 \in V(t) \wedge pos(v_0) = \vec{0}\}$ The **Border Nodes**, that grow along the axes: $V_B(t) = \{v_B|\ v_B \in V(t) \wedge pos(v_B) = m * \vec{j_i}\ \forall \vec{j_i} \in J_+\}$ The **Inner Nodes**, that are all nodes which are neither Root Nodes nor Border Nodes: $V_I(t) = \{v_I|\ v_I \in V(t) \setminus (V_0(t) \cup V_B(t))\}$

A node that wants to accept a new node, must fulfill at least one growth-rule. The node-type specifies which rules can be used to determine if it can grow, and in which directions it can grow. There are three growth-rules. For a given node they specify a set of directions into which this node could grow according to the used rule. All rules contain the basic condition $N(v, \vec{j}, t) = 0$, which ensures that nodes only can grow into directions$\vec{j}$ that are currently free. Root Nodes use the Root-Growth-Rule L_0 to determine into which directions they could grow:

Root-Node-Growth-Rule L_0: Root Nodes can grow into any free positive direction: $L_0(v_0 \in V_0(t)) = \{\vec{j_k} \in J_+|\ N(v_0, \vec{j_k}, t) = 0\}$. **Border-Node-Growth-Rule L_B:** A Border Node v_B can grow along its positive axis, if the position into which it wants to grow is free, and if it has a neighbor, which is an Inner Node: $L_B(v_B \in V_B(t)) = \{\vec{j_k} \in J_+|(N(v_B, \vec{j_k}, t) = 0) \wedge (\exists w \in V(t), \vec{j_l} \in J_+ \setminus \{\vec{j_k}\}$ with $pos(w) = pos(v_B) + \vec{j_l})\ \}$ **Inner-Node-Growth-Rule L_{IB}:** A Border Node v_B or an Inner Node v_I can grow, if the position in direction $\vec{j_i}$ is free, and if the new position has a neighbor in at least one further dimension, which has a common neighbor with the growing node: $L_{IB}(v_{IB} \in V_I \cup V_B) = \{\vec{j_k} \in J_+|(N(v_{IB}, \vec{j_k}, t) = 0) \wedge (\exists w, z \in V(t), \vec{j_l} \in J \setminus \{\vec{j_k}, -\vec{j_k}\})|\ (pos(w) = pos(v_{IB}) + \vec{j_l}) \wedge (pos(z) = pos(v_{IB}) + \vec{j_l} + \vec{j_k})\ \}$ Following these rules it is possible that a node can accept multiple new nodes at the same time. All possible grow directions for a node $v \in V(t)$ are given by: $L(v) = L_0(v) \cup L_B(v) \cup L_{IB}(v)$.

It is possible (and for neighboring nodes even likely) that two different nodes v, w want to grow to the same position at the same time: $pos(v) + \vec{l_j} = pos(w) + \vec{l_k},\ \vec{l_j} \in L(v),\ \vec{l_k} \in L(w)$. This must be avoided, since it leads to overlapping, inconsistent structures. The way to do this, is to provide a locking-mechanism. Nodes can be locked. A lock is related to a certain growth-position. If a node wants to grow into a certain direction, it first has to lock all nodes that belong to the neighborhood of the new position. If at least one node is already locked by another growing node, the growth process must be canceled. This locking-mechanism is not part of the mathematical model described here. The simulation uses a simplified way to avoid those situations by taking advantage of having a global view to all nodes that want to grow. See section 3.4 for further details.

Once a node $v \in V(t)$ with a join-request from a new node v_N has a rule which allows it to grow into a direction $\vec{j_i} \in L(v)$ of an unlocked environment, it will initiate the growth process. The first step of this process is to initiate the new node v_X by giving it a position within the grid: $pos(v_X) := pos(v) + \vec{j_i}$. v_X has to be added to the graph $G(t+1)$'s vertex set. The edge set is appended by two new tuples since all neighbor-links are bidirectional: $V(t+1) := V(t) \cup \{v_X\}$, $E(t+1) := E(t) \cup \{(v, v_X), (v_X, v)\}$. The neighbor-links between v and v_X have to be established:$N(v, \vec{j_i}, t+1) := v_X$, $N(v_X, -\vec{j_i}, t+1) := v$.

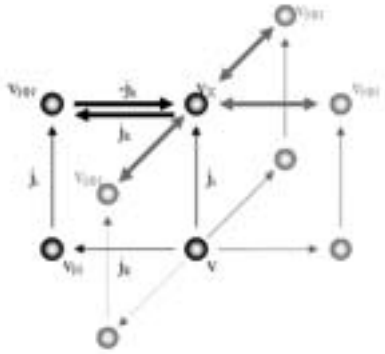

Figure 2: Illustration of the neighborhood-update in a 3d-lattice for a new node v_X, that was accepted by node v in direction $\vec{j_i}$

Now, the neighbor-links of all nodes in v_X's neighborhood have to be updated. To make v find these nodes, all neighbors v_N of v in all dimensions, except that given by $\vec{j_i}$ are requested to make their neighbors v_{NN} in direction $\vec{j_i}$ being a neighbor of v_X (see Fig. 2):
$\forall \vec{j_k} \in J \setminus \{\vec{j_i}, -\vec{j_i}\} | (v_N := N(v, \vec{j_k}, t) \neq 0) \ \wedge (v_{NN} := N(v_N, \vec{j_i}, t) \neq 0) :$
$N(v_{NN}, -\vec{j_k}, t+1) := v_X, \ N(v_X, \vec{j_k}, t+1) := v_{NN}$
$E(t+1) := E(t) \cup \{(v_{NN}, v_X), (v_X, v_{NN})\}$

Summarizing, a node that got a join-request performs the following steps:

- check the rule(s) applicable for this node type.
- if there's no direction to which to grow, reject or forward join-request, else choose one possible direction.
- try to lock the environment, which would be involved in the growth-process
- if locking failed, choose another direction and try again.
- if an unlocked environment could be found, lock it. Iif there's no unlocked environment for any grow-direction, reject or forward join-request.
- perform growth-process (see below) and unlock environment.

The algorithm described here does not yet take any node failures into account. It assumes 'perfect' nodes and communication channels. Detailed strategies on how to deal with node failures in real-life networks is part of further work.

3.4 Simulation

For simulating the 2D-algorithm from [BORDER09] P2PNetSim [P2PNETSIM06], a distributed Java-based network simulator was utilized. A protocol suite for node-(un)locking, neighborhood establishment, and join requests was implemented. It could directly be used in real network applications.

To simplify performance- and efficiency analysis a much more compact, non-distributed Java application was provided for the n-dimensional border-growth implementation. It focuses on the algorithm itself, rather than on "real-life's" technical issues. Since the implementation was intended to be a proof-of-concept, the focus of the implementation was to provide a quick, stable realization of the algorithm. The implementation makes use of the fact, that there is a global view on the whole network in the implementation: No local locking mechanism was implemented. Instead, in each discrete time-step t a data structure is built that contains lists of those nodes, that want to grow to the same position. From this list, only one randomly selected node is allowed to grow. From the 'local point of view', this would be the node that first got the chance to lock the new position's environment. The advantage is, that the simulation needs fewer resources and can simulate growth-processes with millions of nodes on a single machine. Comparisons between this code running with two dimensions with the 2d-simulation from [6], which uses (un-)locking-protocols show that there are no qualitative differences between the global and the local way of resolving conflicts. Later versions will be distributed again, and will implement the lock-/unlock-protocol again in order to get closer to a real networking scenario.

Fig. 3a, 3b, 3c, and 3d show the output of four small simulations with 40 nodes that grew in two, three, four and five dimensions. Fig. 3e shows the output of a larger simulation that built 20.000 nodes into a five-dimensional lattice. Remember, that, for simplicity, the algorithm described here, just considers the positive quadrants. The dark lines are the five positive axes of the lattice - projected to a two-dimensional circle.

4 Simulation results

Fig. 4 shows the growth-rates in the first 100 time steps for simulations with two, three, four, and five dimensions. In the simulation of two dimensions the growth-rate has roughly linear characteristics (notice, that the growth-rate axis has a logarithmic scale.) After 100 time steps the growth-rate is 17 nodes/time-step. The five dimensional case shows an exponential course which leads to a growth-rate of about 300.000 nodes/time-step after 100 time steps. As expected, the growth-scalability can be strongly improved by increasing the structure's dimension. Assuming that the number of nodes and the growth rate follow these equations: $\#nodes = a_{\#nodes} * t^{n_{\#nodes}}$ and $growthRate = a_{growthRate} * t^{n_{growthRate}}$ The coefficients a and exponents n were computed, based on the simulation data in Fig. 4, as follows: The tables in show that growth-scalability is improved exponentially with the increasing dimensions, though the fractions of growable nodes decrease with more dimensions. The nodes' ability to grow into several dimensions within one time step

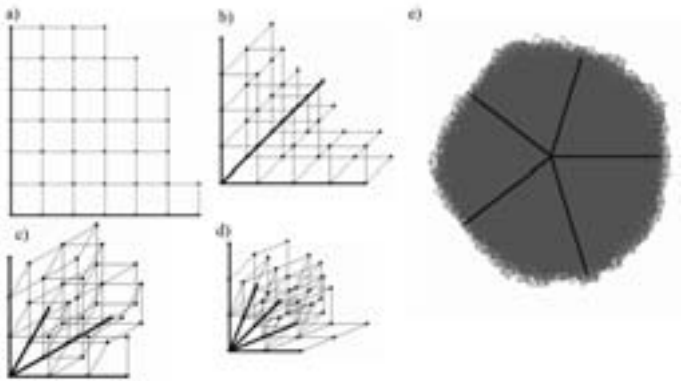

Figure 3: a-d: simulation with 40 nodes in a 2d-, 3d-, 4d- and 5d-lattice, e: simulation with 5 dimensions and 20k nodes

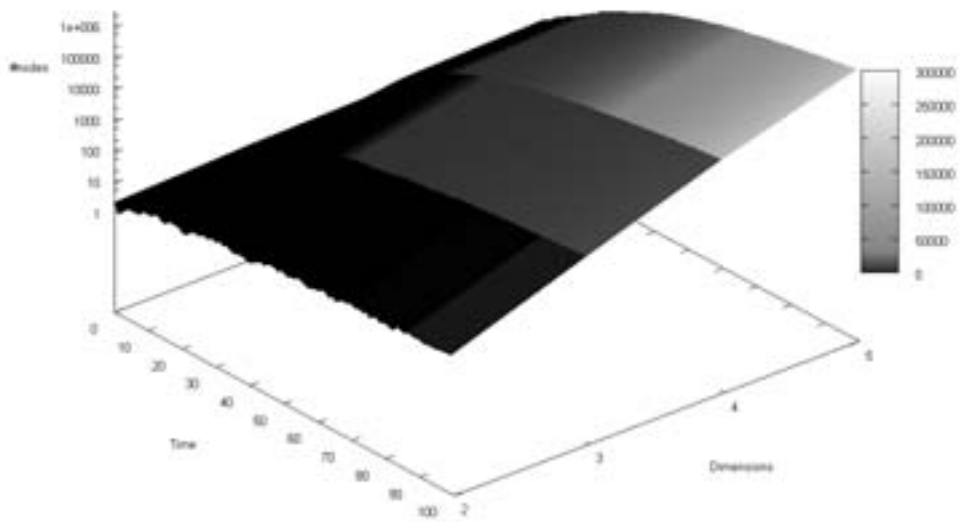

Figure 4: a: growth rate depending on time and dimension

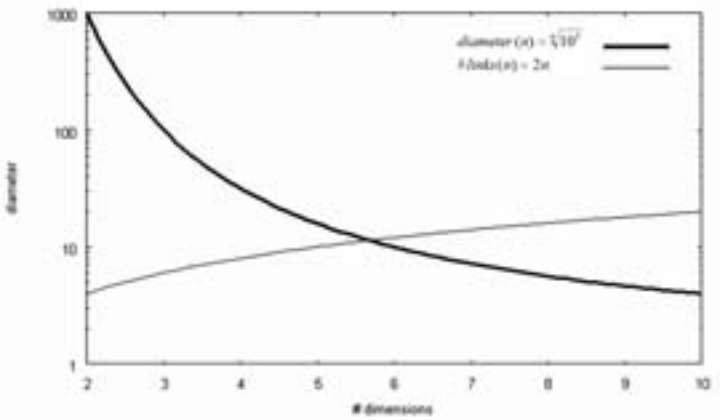

Figure 5: The diameter (bold graph) and the number of links per node (thin graph) of a network with 1.000 nodes depending on the number of dimensions

	2d	3d	4d	5d
a	0,269	0,087	0,025	0,006
n	1,912	2,869	3,807	4,736

	2d	3d	4d	5d
a	0,563	0,470	0,303	0,169
n	0,892	1,729	2,548	3,348

Figure 6: left table: growth equation parameters, right table: growth-rate-equation parameters

does obviously not affect the growth-behavior in a positive way. Detailed analysis of the growth-behavior with increasing dimensions will be subject of further research.

For estimating the network's diameter it is assumed as a first approximation that the structure has a n-dimensional cubic shape, that grow uniformly into each possible direction. The diameter of a network with 1000 nodes depending on the dimension then would be: $diameter(n) = \sqrt[n]{1000}$ (see bold graph in Fig. 5). While the network's diameter decreases exponentially with the number of dimensions, the number of neighbor-links that have to be managed by each node, grows only linearly: $\#links(n) = 2n$ (thin graph in Fig. 5).

By providing more than two dimensions a node has multiple directions in which it could grow within a single time step. Originally it was expected that this will improve growth-behavior especially in higher dimensions. Since this is valid for all nodes, this leads to the situation that more nodes compete for the same position, which lowers the advantage of the nodes' possibilities of multi directional growth. That might be the reason, why the algorithm can't really take advantage of this effect. This might change in much later time steps of the simulation, especially when higher dimensions (> 7) are used.

Beside that there are other factors that hinder growth, the more so as more dimensions are used. Detailed investigations have to be made to find exact formulas to predict the n-dimensional growth behavior.

5 Conclusion & Outlook

By increasing the structure's dimension, the scalability can be significantly improved. The structure's diameter decreases exponentially. The complexity of the growth-process stays constant independently of the number of dimensions, the complexity of the update-process of a new node's environment increases linear with the number of dimensions. Since there is no limit for the number of dimensions, topologies with small diameters can be established. [RFID07] introduces an algorithm that uses EPCs (Electronic Product Codes) to manage product information in a decentralized network spanning across multiple organizations. The two-dimensional address-space of such a network is defined by splitting a 28-bit part of the EPC into two components representing the coordinates in the network. It could be improved by mapping the 28-bit code to a, for example, seven-dimensional border-growth-structure. This would result into a network with a diameter of $2^{\frac{28}{7}} = 16$.

References

[LCP04] *Lua, E.; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S.*: "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", IEEE Communications Survey and Tutorial, March 2004.

[SUL04] *Sakarian, G.; Unger, H.; Lechner, U.*: "About the value of Virtual Communities in P2P networks.", Proc. ISSADS 2004, Guadalajara, Mexico, Lecture Notes in Computer Science (LNCS) 3061, Guadalajara, Mexico, 2004.

[RFH01] *Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R. M.; Shenker, S.*: "A Scalable Content Addressable Network", Proc. ACM SIGCOMM 2001.

[CHORD01] *Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M. F.; Balakrishnan, H.*: "Chord: A scalable peer-to-peer lookup service for internet applications.", In Sigcommâ01, Proc. of the 2001 conference on Applications technologies architectures and protocols for computer communications, pp. 149-160.

[STRUCT02] *Unger, H.; Unger, H.; Titova, N.*: "Structure Building in Distributed Communities", A. Tentner (ed.) "High Performance Computing (HPC) '2002",San Diego, 2002

[BORDER09] *Berg, D.; Unger, H.; Sukjit, P.*: "Borderline-growth a new method to build complete grids with local algorithms", 2nd International Workshop on Nonlinear Dynamics and Synchronization 2009 (INDSâ09), S. 95-99, Klagenfurt, Austria, July 2009,Shaker Verlag, Aachen, 2009

[RFID07] *Berg, D.; Coltzau, C.; Sukjit, P.; Unger, H.; Nicolaysen, J.*: "Passive RFID tag Processing using a P2P architecture.", Malaysian Software Engineering Conference 2007, S. 169-179, 2007

[THERM09] *Lertsuwanakul, L.; Unger, H.*: "A Thermal Field Approach in a Mesh Overlay Network.", 5th National Conference on Computing and Information Technology (NCCIT'09),Bangkok, Thailand, 2009

[3DIOS09] *Coltzau, C.; Unger, H.*: "3DIOS â Konzept eines Internet Operating Systems.", Gemeinschaften in Neuen Medien (GeNeMe) '09, 2009

[P2PNETSIM06] *Coltzau, C.*: "Specification and Implementation of a Simulation Environment for Large P2P-Systems", Diploma, University Of Rostock, 2006