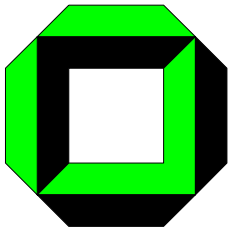
Lanes – A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks

Michael Klein Birgitta König-Ries Philipp Obreiter
{kleinm | koenig | obreiter}@ipd.uni-karlsruhe.de

May 19, 2003

Technical Report Nr. 2003-6



University of Karlsruhe
Faculty of Informatics
Institute for Program Structures and Data Organization
D-76128 Karlsruhe, Germany

Abstract

The ability to discover services offered in a mobile ad hoc network is the major prerequisite for effective usability of these networks. Unfortunately, existing approaches to service trading are not well suited for these highly dynamic topologies since they either rely on centralized servers or on resource-consuming query flooding. Application layer overlays seem to be a more promising approach. However, existing solutions like the Content-Addressable Network (CAN) are especially designed for internet based peer-to-peer networks yielding structural conditions that are far too complex for ad hoc networks. Therefore, in this paper, we propose a more lightweight overlay structure: lanes. We present algorithms to correct and optimize its structure in case of topology changes and show how it enables the trading of services specified by arbitrary descriptions.

1 Introduction

Multihop ad hoc networks (MANETs) are networks of mobile devices that communicate with one another via wireless links without relying on an underlying infrastructure. This distinguishes them from other types of wireless networks, e.g. cell networks. In order to achieve communication, each device in a MANET acts as endpoint and as router forwarding messages to devices within radio range.

MANETs are a sound alternative to infrastructure based networks whenever an infrastructure has never been available, is no longer available, or cannot be used. They are also attractive in situations where it just seems more natural to communicate directly among devices that are in physical proximity instead of relying on remote infrastructure components. Example applications for the first cases are, e.g., military applications, disaster relief, and communication between vehicles. Examples for the latter are conferencing or e-learning applications and personal area networks.

Over the last few years, much effort has been put into making MANETs become a reality. Research has been focused on the development of appropriate routing protocols, methods for energy preservation, and other issues on the lower four ISO/OSI layers. While this work is still ongoing, by now, a sound technical basis for MANETs exists. In our opinion, it is thus time to start thinking about how to support applications based on MANETs. Just as the mere fact that computers were networked was not sufficient to allow for transparent access to distributed resources, the mere fact that it is technically possible to form a MANET is not sufficient to allow for effective usage of the resources contained within these networks. We believe that the main prerequisite for such usage is the ability to advertise and find services. Examples for services range from the ability to print documents, to the lending of computing power, to the processing of database queries, to the delivery of documents or more generally information, to navigation services, and to the usage of specialized technical equipment. With this ability it becomes possible to use the distributed knowledge, computing power, and capabilities spread throughout a MANET.

Unfortunately, existing mechanisms for service discovery are not well suited for MANETs, since they either rely on central directory servers – a component the existence of which cannot be guaranteed in a MANET – or they produce a huge message overhead – an approach that is not feasible in resource-constrained networks. More sophisticated approaches analyze the content of the service requests to route them semantically. In general, they typically support rather primitive service descriptions only. We believe that for an efficient usage of services in MANETs (or for that matter in any network), service discovery based on semantically rich, ontology based service descriptions needs to be supported.

In this paper, we thus propose a new method for service advertisement

and discovery in MANETs. The basic idea is to define a two-dimensional overlay structure, called *lanes*, which is similar to, but less strict than the one introduced in the approach of the Content Addressable Network (CAN) [1]. One dimension of this overlay is used to propagate service advertisements, the other one to distribute service requests. This results in a fault-resilient and efficient structure, which can be used for semantics-based service discovery. While we are working on developing appropriate service descriptions, the description language is well beyond the scope of this paper. Furthermore, the approach described here is completely independent of the concrete service description used.

The remainder of the paper is organized as follows: In Section 2, we discuss existing approaches for service trading and explain why they are not usable in MANETs. Section 3 introduces CAN and discusses how its structure can be used for service advertisement and discovery, but also why it cannot be used "as is" in MANETs. Thus, we then introduce in Section 4 a more lightweight overlay structure, called lanes, by softening some of the conditions CAN is based on, and describe in detail how this overlay can be used for service discovery and advertisement. The paper ends with a conclusion and an outlook to future work in Section 5.

2 Related Work

Service discovery for distributed environments is often designed for internet based networks and thus does not take the characteristics of mobile ad hoc networks into account. Generally, we can distinguish four main approaches: central service repositories, flooding of requests, hashing, and semantic routing. In this section, we will examine these techniques in details, show that existing discovery protocols fit into one or more of these categories, and explain why most of them are not suitable in ad hoc environments.

2.1 Centralized Approaches

In centralized architectures, we have one or a few dedicated central devices communally storing the descriptions of all services offered in the network. Either each client knows all these servers or (more commonly) the servers are federated, i.e., they know how to communicate with one another. Therefore, in general, service offerers proactively advertise their services to one of the servers, whereas searchers contact one server to query for suitable services. To keep the repositories up to date, obsolete services have to be signed off manually or are removed periodically.

It is obvious that these architectures are not well suited for mobile ad hoc networks as no server could guarantee its availability because of frequent topology changes. Only in extremely stable networks or network regions, central approaches might be applicable. Nevertheless, in other domains,

these architectures have been widely accepted. For example, CORBA's Trading Object Service [2], Jini [3], and Bluetooth [4] use centralized mechanisms. This is also true for protocols like Napster [5] for file sharing or the more sophisticated Service Location Protocol SLP [6]. Furthermore, web services, which are getting more and more important in business applications, are traded in central directories via the Universal Description, Discovery, and Integration of Web Services (UDDI) [7]. Also, approaches for service search in agent environments (like the lightweight frameworks LEAP [8] and MircoFIPA-OS [9]) mainly rely on central components.

2.2 Service Discovery by Flooding

In contrast to centralized directories, in a broadcasting architecture, service offerers do not distribute their service descriptions onto other nodes in the network, but leave them stored on their own device. This leads to a reactive service trading: service requests have to be forwarded to all members of the network where they are compared with the stored descriptions. To do that, a device interested in a special service typically sends its search message to all reachable nodes. If one or more of these nodes can satisfy the request, a response is sent back to the requestor. In any case, the search message is forwarded to all reachable nodes except for the previous sender. To reduce duplicate node querying, sophisticated flooding algorithms have been proposed.

Generally, these broadcasting mechanisms are not suited for mobile ad hoc networks, either, due to their heavy consumption of bandwidth and energy, which are not unlimitedly available on mobile devices. Nevertheless, in regions with extremely high dynamics, broadcasting could be the only possible technique. Typical representatives of this technique are the Simple Service Discovery Protocol (SSDP) [10], completely decentralized file sharing protocols like Gnutella [11] and JXTA-Search [12].

2.3 Hash-Based Approaches

Another technique besides server based or broadcast oriented approaches is hashing. Generally, a hash function is used to transform a given service description into a numerical value. This value can be transduced into an address of a device in the network. Service announcements as well as service requests are sent to and compared by this calculated device.

The main problem of hashing approaches originates from their mathematical characteristics: As they map values randomly all across the network, in general, semantical closeness of service descriptions is not represented in physical closeness of addresses. Therefore, hashing can only be used for service descriptions with very little semantics (e.g., Boolean terms of keywords). Semantically rich descriptions (e.g., ontology based approaches like DAML-

S [13]) cannot be processed reasonably by hash functions as a requestor will rarely find exactly matching results and thus is merely interested in finding similar services. The most prominent hashing approaches are OceanStore [14], Globe [15], Chord [16], Freenet [17], Tapestry [18], and CAN [1]. We we will examine the latter one more deeply in the next section.

2.4 Semantic Routing

A technique that is more adapted to ad hoc networks is semantic routing. Taking the pure flooding algorithm as starting point, each node tries to minimize the amount of forwarded request messages by not sending them to all known neighbors, but intelligently choosing only a few of them by inspecting the semantics of the request. On the one hand, this can be achieved by building and maintaining an application-layer overlay reflecting the offered services (as in the Intentional Naming Scheme (INS) [19], the agent-based discovery framework Allia [20], or our Multi-layer Clusters [21] and Service Rings [22]). On the other hand, each node can base its forwarding decisions on collecting information about previously sent messages (as in NeuroGrid [23] or in the Routing Indices approach [24]). Generally, these approaches are a step in the right direction helping to solve our problem, but either rely on internet-based peer-to-peer systems, are only able to handle simple descriptions consisting of attribute-value pairs, or need a huge amount of messages to learn good routing rules.

2.5 Conclusions

With respect to our goal, i.e. a semantics based service discovery in MANETs, the main problem of the approaches presented in the previous sections relates to their architecture to support service discovery. Generally, this architecture is not suitable for an application in highly dynamic networks (in the case of server based architectures) or assumes the existence of large bandwidths (in the case of message broadcasting approaches), which are not available in most mobile devices because of their limited resources. Hash-based architectures, on the other hand, only support semantically weak service descriptions and therefore are not able to fulfill our needs. The presented concepts for semantic routing are a step in the right direction, but are not constructed for our demands.

3 Using the CAN Overlay Structure for Service Trading

3.1 Basic Idea of CAN

When examining existing approaches to service discovery in mobile ad hoc networks in the previous section, we noticed that they only insufficiently mediate between the user's requested functionality (i.e. efficient, semantic service trading) and the characteristics of a mobile ad hoc network (highly dynamic topology and weak device capabilities). Thus, it seems reasonable to insert an additional layer above the transport layer (Layer 4) that bridges these two parts. To achieve this, the new layer should build, maintain, and offer an overlay structure

- that is constructed to optimally serve the mechanisms of service trading (i.e., announcement and discovery of service descriptions) *and*

- that can be efficiently adapted to the constantly changing topology of the underlying network.

The structure used in RATNASAMY et al's *Content Addressable Network* (CAN) [1] offers a good starting point for an overlay structure that can be used to gain the above-mentioned properties. For CAN, a virtual d-dimensional address space is defined and laid on top of the nodes of the network. It helps to set up a distributed hash-table that can be used to store and re-access arbitrary (key, value) pairs in the network. CAN was designed for internet-based peer-to-peer networks, which are similar to ad hoc networks to some extent. In the next sections, we will give a short introduction to CAN, explain how its structure can be used to implement semantic service trading, and examine whether its structural conditions can also be asserted in mobile ad hoc networks.

3.2 A Short Introduction to CAN

The *Content Addressable Network* CAN [1] was designed to offer the functionality of a distributed hash table in a large peer-to-peer network. To achieve this, CAN forms an application-layer overlay that is derived from a virtual d-dimensional Cartesian coordinate space D. This space D (e.g., $[0,1] \times [0,1]$ for $d = 2$) is completely abstract and is not correlated to any physical location. It just helps to define the conditions that have to be fulfilled to form a valid CAN overlay. Therefore, we have the following definition:

Definition 1 [Valid CAN Structure]
Let D be a d-dimensional axis-parallel hypercuboid. An network forms a valid CAN overlay if the following conditions are fulfilled:

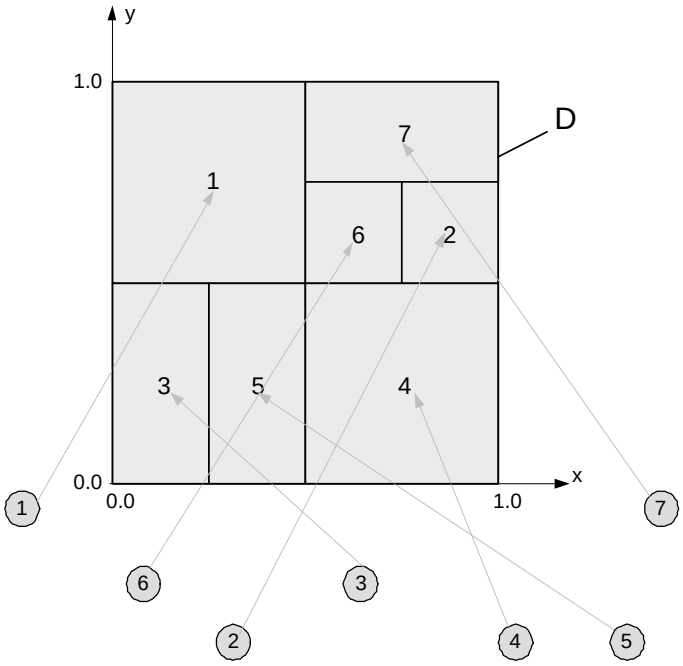


Figure 1: Example of a 2-dimensional CAN structure. The space $D = [0, 1] \times [0, 1]$ is distributed among 7 nodes without any correlation to their physical location. The rectangle a node owns defines the neighbors it must be aware of. In the example, Node 5 has to know the addresses of Node 1, 3, and 4, because their rectangles are adjacent to its own.

1. *Each node in the network "owns" one nonempty subset*[1] *of D in the shape of a d-dimensional, axis-parallel hypercuboid (e.g., a rectangle in 2-dimensional space).*

2. *All these distributed hypercuboids are disjunct and their union exactly yields the complete space D.*

3. *Each node knows the IP addresses of all other nodes that own hypercuboids being adjacent to its own hypercuboid.*

An example of such a two-dimensional space and its distribution among 7 nodes can be seen in Figure 1. Each node (depicted by numbered circles) owns a rectangle within the area $D = [0, 1] \times [0, 1]$. This rectangle defines the neighbors a node has to know. Therefore in the example, Node 5 must be aware of the Nodes 1, 3, and 4.

This distributed space helps to determine the device that has to store a certain (key, value) pair. More precisely, the key is transformed by a

[1] While repairing structural errors, for a short time, two or more adjacent subsets are possible.

network wide known, deterministic hash function h into a coordinate point $x = (x_1, x_2, \ldots, x_d) = h(key) \in D$. The device owning the cuboid which contains this point is responsible for filing the (key, value) pair. To access this node, a request to the node must be routed through the CAN overlay, which can simply be done with the help of a greedy algorithm: Every node forwards the packet to that one of its neighbors whose cuboid is closest to the destination coordinate x. Therefore, the average path length of a arbitrary routing request is in $O(\sqrt[d]{n})$, where n is the number of nodes in the network.

CAN has to deal with several situations that can violate the above-mentioned conditions of the overlay structure: a new node enters the net, a node leaves the net intentionally or a node is not reachable anymore (because it has left unintentionally or the network has been partitioned).

Node entrance. When a new node N enters the net, Condition 1 is not fulfilled anymore because every node has to own a *nonempty* hypercuboid in the space. One possibility to restore this is to transfer half of another node's cuboid to N and to adjust N's neighboring links according to Condition 3. More specifically, five steps are performed: (1) N randomly chooses a point $x = (x_1, x_2, \ldots, x_d)$ from D. (2) N sends a split request to the Node X responsible for x via the normal CAN routing algorithm. (3) X divides its cuboid and sends the half containing x together with the appropriate (key, value) pairs and the information of the neighboring nodes back to N. (4) N receives the cuboid and updates its neighbor links. (5) Both N and X inform their neighbors about the new conditions.

Intended Node departure. When a node N leaves the net, Condition 2 is violated because the union of all cuboids does not result in the entire space D. Therefore, it must be ensured that N's cuboid is transferred to another, still active node. This is done in the *handover algorithm*: (1) N chooses the one of its neighbors (say X) that can best handle its cuboid. (2) N transfers its cuboid together with the stored (key, value) pairs and the information of its neighbors to X. (3) X merges its two cuboids and adjusts its neighbors. If the resulting space is not a regular hypercuboid, a background process tries to redistribute one of the overlapping parts (see *Background Zone Reassignment Algorithm* in [1] for more details).

Node unreachability. Generally, this problem can be solved by the handover algorithm from above. In most cases, it is more difficult to determine whether a node is absent from the network[2]. To do that, in CAN, each node checks the status of its neighbors by periodically sending update messages containing its own cuboid, its neighbors and their cuboids to all

[2]The reason for unreachability is basically undeterminable so that unintended node departure and network partition have to be handled in the same way.

of its neighbors. If such a message is absent for a certain time, the neighboring nodes assume the node's failure and one of them starts the handover algorithm to reestablish Condition 2.

The virtual address space provided by the overlay of CAN can be used for more than just storing (key, value) pairs. In [25], RATNASAMY et al. present an efficient application-level multicast profiting from the regular structure of CAN.

3.3 Application of a 2-dimensional CAN Structure to Service Discovery

In this section, we want to show how CAN's overlay structure can be used for service trading based on a semantically rich service description. However, the rich semantics prevents us from simply using CAN's hashing mechanism to determine the storage node for a service description because the employed hash functions only preserve semantic similarity in case of very simple descriptions (e.g., keywords combined by Boolean operators, see [1]). Nonetheless, if we want to support complex service descriptions, which might even be based on ontology languages like DAML-S [13], it is not possible to use their semantic content for determining the storage location. Therefore, we have to separate the overlay structure from the concrete description and build it by exclusively using the fundamental semantics of service trading, only, i.e. there are two orthogonal dimensions: one for announcing offered services and one for searching suitable services. These two aspects can be directly described by a 2-dimensional CAN overlay (so $d = 2$). The first dimension (in the following the y axis) defines the direction and the devices where service offers have to be stored, the second dimension (in the following the x axis) defines where services have to be searched.

In the following, we want to examine the detailed algorithms of the two tasks:

Algorithm 1 [Service announcement]
Node N wants to offer service s.

1. *N chooses an arbitrary point $p = (x_0, y_0)$ from its rectangle.*
2. *N forwards the service description together with its own address to those two neighbors[3] whose rectangles are intersected by the straight line $x = x_0$.*
3. *Each node receiving the service description stores it and sends it to that opposite neighbor owning a rectangle that is intersected by the straight line $x = x_0$ (if existent).*

[3] marginal nodes forward it to one or zero neighbors only

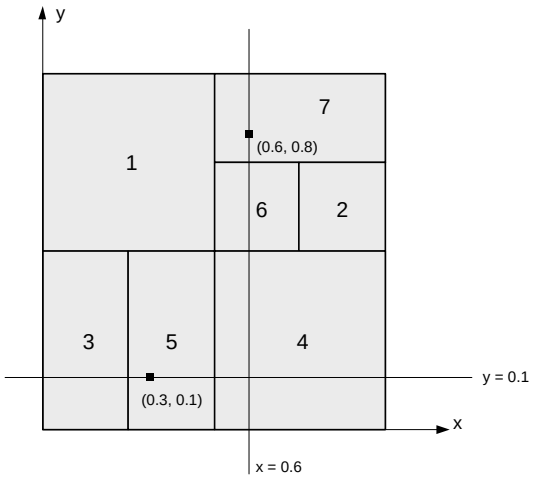


Figure 2: Generally, service offers are distributed in y direction, services are searched in x direction. In the example, Node 7 wants to offer a service and chooses point $(0.6, 0.8)$ from its rectangle. Thus, the description is forwarded to and stored on any node that owns a rectangle that is intersected by the straight line x =0.6, here Nodes 6 and 4. Node 5 wants to search for a service and chooses point $(0.3, 0.1)$ from its rectangle. Therefore, the search request is forwarded to those nodes that own a rectangle that is intersected by the straight line $y = 0.1$, here Node 3 and Node 4. On its way, the request can be compared to every service offer description in the network.

In Figure 2, Node 7 wants to offer a service and chooses point $p = (0.6, 0.8)$ from its rectangle. As its neighboring Node 6 owns an area which is intersected by the straight line $x = 0.6$ the service description is sent to that node, which stores it and then forwards it to its opposite neighbor, Node 4. Since this is a marginal node, the offer is stored there, but not forwarded any further.

On average, this algorithm distributes the descriptions to $\sqrt{n}$ nodes where n is the total number of nodes in the network.

Algorithm 2 [Service search]
Node N wants to search for a device offering s or a similar service.

1. *N chooses an arbitrary point $p = (x_0, y_0)$ from its rectangle.*
2. *N forwards its service request together with its own address to those two neighbors whose rectangles are intersected by the straight line $y = y_0$.*
3. *Each node receiving the service request checks whether it stores a matching service description. If yes, it sends a success message back to Node N. If not, it forwards the message to that opposite neighbor owning a rectangle that is intersected by straight line $y = y_0$ (if existent).*

In Figure 2, Node 5 starts a search for a service and chooses point $p = (0.3, 0.1)$ from its rectangle. As its neighboring Nodes 3 and 4 own areas that are intersected by the straight line $y = 0.1$, the request is sent to these nodes and the latter compare it with their stored descriptions. If the service searched for by Node 5 is the one that Node 7 offers, a match is achieved in Node 4. In this case, the address of Node 7 is sent back to the requestor 5.

3.4 Guaranteeing the Structural Conditions in MANETs

In Section 3.2, we presented techniques for guaranteeing the integrity of the three CAN conditions even in case of networks changes in the underlying layer. As these algorithms are specially designed for the characteristics of peer-to-peer networks, it is questionable whether they can be transferred to mobile ad hoc networks without any change. One problem arises, for example, in case of node entrance: The algorithm to log into the CAN structure does not try to find an optimal neighborhood for the new node, but chooses arbitrary neighbors resulting in the split of the hypercuboid of a randomly chosen node in the network. Moreover, this neighborhood is not optimized later leading to increasingly inefficient links that are not aligned with the physical network topology. In [1], the authors present a method to attenuate these effects by introducing commonly known landmark nodes. The tuple of distances to these landmarks indicates an area of similarly located nodes. Nodes from this area should be preferably chosen as the new node's neighbors resulting in a higher adaption of the CAN structure to the network topology. However, this method is not applicable to ad hoc networks because of the lack of fixed, commonly-known nodes.

Another problem arises from the detection of unreachable nodes. In CAN, this is achieved by relatively extensive ping messages, which each node sends to all of its neighbors (normally at least four). For ad hoc networks, this is far too expensive, as it requires too much of the rare bandwidth.

Generally, the strict grid structure of CAN seems to hamper the transference to mobile ad hoc networks. Thus, the idea is to weaken the structural conditions of the CAN overlay so that they a) are applicable to MANETs and b) still offer the possibility to efficiently trade services. Such a structure is presented in the next section.

4 A Lightweight Overlay for Service Discovery in Ad Hoc Networks

4.1 Basic Idea

As we have seen in the previous section, the structural conditions of CAN cause some severe problems in highly dynamic MANETs with resource limited devices because of the strict grid organization, in which each node

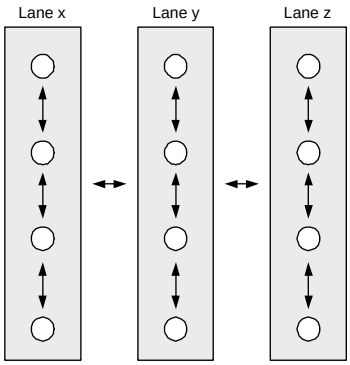


Figure 3: Within a lane, nodes are fixedly ordered and each of them knows its predecessor and its successor. The lanes themselves are loosely coupled: Nodes in the same lane share the same anycast addresses which help to use anycast routing for sending messages from lane to lane.

(except for marginal nodes) has to maintain at least 4 links to neighboring devices (for $d = 2$). For MANETs, it is therefore imperative to weaken these conditions. This could be possible because of the fact that for service discovery the query only needs to be passed to a set of devices so that the x intervals of their owned rectangles cover D's complete x interval. The y coordination does not matter at all. Thus, request messages do not necessarily have to be in line with their searching device (i.e., the service searcher and its request need not be on devices with rectangles containing the same y coordinate). This allows to abolish the fixed assignment of nodes in x direction, i.e. there are only *lanes* of nodes, which are loosely coupled. Within a lane, there is still a strict relationship of predecessors and successors and also the lanes are arranged in a well defined order. From the outside, all nodes within one lane can be treated equally, if sending a service request to that lane – any arbitrary node in it has full information to handle the request. As a consequence, *anycast routing* can be used to send messages from lane to lane when nodes belonging to the same lane share the same *anycast address* (see [26] for more details on anycast addresses). Figure 3 gives an outline of this idea.

In the following section, the formal conditions of the lane structure are introduced. After that, we will show the mechanisms to trade services on top of such a structure. The algorithms that guarantee the correctness of the structure are presented in the subsequent section.

4.2 Formal Conditions of Lanes

Analogously to CAN, we define a valid Lanes structure:

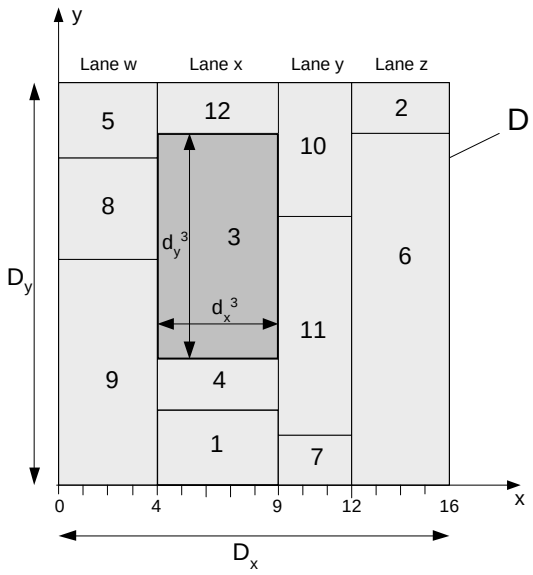


Figure 4: A correct lane structure. Node 3 owning the highlighted rectangle has to know the addresses of Node 4 and 12 because their rectangles are adjacent in y direction. Like any other node in Lane x, Node 3 has the two anycast addresses 4 and 8 resulting from its interval $d_x^3 = [4, 9)$.

Definition 2 [Valid Lanes Overlay]
Let $D = D_x \times D_y = [d_{x0}, d_{x1}) \times [d_{y0}, d_{y1})$ be a rectangle with $d_{x0}, d_{x1}, d_{y0}, d_{y1} \in \mathbb{N}$, i.e. its boundaries are natural numbers. A network forms a valid Lanes overlay if the following conditions are fulfilled:

1. *Each node N in the network "owns" one nonempty rectangle $d^N = d_x^N \times d_y^N = [d_{x0}^N, d_{x1}^N) \times [d_{y0}^N, d_{y1}^N)$ with $d_x^N \subseteq D_x$ and $d_y^N \subseteq D_y$ and $d_{x0}^N, d_{x1}^N, d_{y0}^N, d_{y1}^N \in \mathbb{N}$.*

2. *For each two nodes N and K we have: d_x^N and d_x^K are equal (so the two nodes are in the same lane) or disjunct (so they are in different lanes).*

3. *All these distributed rectangles d^N are disjunct and their union exactly yields the complete space D.*

4. *If d^N and d^K are adjacent in y direction (i.e. $d_{y1}^N = d_{y0}^K$ or $d_{y0}^N = d_{y1}^K$), then node N knows the address of node K. We denote N's upper neighbor with $N.T$ and its lower one with $N.B$, if existent.*

5. *Each node N owning a rectangle with $[d_{x0}^N, d_{x1}^N)$ is addressable by the two anycast addresses d_{x0}^N and $d_{x1}^N - 1$. Therefore, neighboring lanes can be addressed by $d_{x0}^N - 1$ (if $\neq d_{x0} - 1$) and d_{x1}^N (if $\neq d_{x1}$).*

An example of such a lane structure is depicted in Figure 4: The area D is completely divided among the 12 nodes so that no two areas overlap and each two d_x^N intervals are equal or disjunct (resulting in 4 lanes). Consider the highlighted rectangle which is owned by Node 3. As it is adjacent in y direction to rectangles owned by the Nodes 4 and 12, Node 3 has to know their addresses. From its interval $d_x^3 = [4, 9)$ we can derive the two anycast addresses 4 and 8, which are shared by all members in Lane x. Therefore, the neighboring Lanes w and y can be addressed by the anycast addresses $4 - 1 = 3$ and 9.

Note that the virtual rectangles directly (in case of the anycast addresses) or indirectly (in case of the unicast addresses within a lane) define the addresses a node has and also has to know. Therefore, analogously to CAN, algorithms changing the overlay structure like lane splitting only need to operate on this virtual space – the corresponding address adaptions are automatically defined by this.

To sum up, we use a combination of a proactive structure within one lane (allowing to use unicasts to well-known predecessors and successors) and a reactive structure between the lanes (leading to anycasts to reach an arbitrary node in neighboring lanes). The following section will show how these different techniques map to the different characteristics of service announcement and service search.

4.3 Service Announcement and Discovery in Lanes

The algorithms for service trading on top of a lanes structure are similar to the ones for a general CAN structure (see Section 3.3). As the announcement of service descriptions is an event whose long-term effects (i.e. the storage of the description on several devices in the network) have to be maintained consecutively by the network (with the help of leases for instance), it seems reasonable to use the proactive, inner lane structure as communication path to distribute the offer descriptions. On the other hand, searching for services is a one-time action, which does not lead to a change that needs to be persisted. Thus, flexible anycast communication between the lanes seems to be a suitable direction of communication. With that knowledge, we can develop the trading algorithms:

Algorithm 3 [Service Announcement]
Node N wants to offer service s.

1. *N sends a* `ServiceOffer` *message containing a description of s and N's address to $N.T$ and $N.B$ (if existent).*

2. *Each node X receiving a* `ServiceOffer` *from its upper node $X.T$/lower node $X.B$, stores it and forwards it to its opposite neighbor $X.B$/$X.T$ (if existent).*

Algorithm 4 [Service Search]
Node N wants to search for a device offering s or a similar service.

1. *N checks whether it has stored an appropriate service description on its own device (stemming from the own lane). If not, N sends a* `ServiceRequest` *message containing the description of s and N's address to the neighboring lanes using anycast routing to the addresses $d^N_{x0} - 1$ (if not $d_{x0} - 1$) and d^N_{x1} (if not d_{x1}).*

2. *Each node receiving a* `ServiceRequest` *checks its service description memory for suitable services in the own lane. If one of the descriptions fits, a* `ServiceFound` *response containing the unicast address of the service offerer is directly sent back to the requestor. If no description fits, the request message is forwarded to the opposite neighboring lane (using anycast addresses like in 2.). If such a lane is not existent, a* `ServiceFailure` *message directly addressed to the service requestor is sent back.*

4.4 Asserting the Structural Conditions in Lanes

In MANETs, similar situations as in peer-to-peer networks can arise that result in violating the structural conditions of the overlay. Due to their high dynamics, in general, ad hoc networks face these problems more often and more severely. Therefore, the correction algorithms have to be more flexible. Analogously to Section 3.2, we consider intended login and logoff of a device as well as broken overlay links. In the following, we show that the flexible lanes structure is well suited for dealing with these problems in ad hoc networks by presenting possible correction algorithms.

4.4.1 Intended Node Login/Logoff

Logging into the network is not very difficult if new nodes are only permitted to join *exactly one existing* lane. Therefore, the entrance of a node neither results in a new lane[4] nor merges two or more existing lanes. However, this might happen in a possible correction step when a large lane consisting of too many nodes is divided into two neighboring lanes or when two short lanes are combined to one new lane (see Section 4.5.2: "Lane Splitting and Merging"). Having said this, we can develop the login algorithm:

Algorithm 5 [Login]
Node N wants to join the network.

1. *N broadcasts a* `LoginRequest` *containing its own address to all nodes it can reach within a single hop.*

[4]One exception is the first node, which initiates the first lane by claiming the entire space D.

2. *Each node X receiving a* `LoginRequest` *sends a* `LoginOffer` *containing its address, the address of its upper neighbor $X.T$, and the length of its lane (if known) back to the requestor.*

3. *N collects these offers and chooses one of them. It prefers offers of a node X if $X.T$ has also sent an offer (resulting in two very efficient one-hop connections for N) or X's lane length is short (helping to minimize lane splittings).*

4. *N sends* `LoginAccept` *messages to the chosen X and its upper neighbor $X.T$.*

5. *X halves its rectangle horizontally and sends a* `LoginConfirm` *message to N. This message contains the upper half of X's rectangle as well as X's stored service descriptions. By that, X and $X.T$ have to update their neighbors to integrate N into the lane.*

6. *When N receives the confirmation, it stores the rectangle and the descriptions and is able to use the benefits of the network structure by offering and/or searching for services.*

As logging off from the network is very simple too, we will not explain that algorithm here for reasons of brevity. The only important aspect to mention is that all offered services of the leaving node are removed with the help of a `OfferRemove` message that is forwarded through the lane. Note that no extensive lease mechanism is needed as each invalid service description is explicitly removed.

4.4.2 Broken Connections Between Nodes

To repair broken links within one lane, we first have to detect them. This is done by periodical `Ping` messages that every node N sends to its upper neighbor $N.T$ containing N's and $N.B$'s address. Generally, if such a message is missing for a certain time, a Node X assumes a broken network connection to its lower neighbor $X.B$ (because $X.B$ unintentionally left the lane or a network partition has occurred). In this case, X tries to contact $X.B.B$ (known from previous ping messages) in order to inform it with a `LaneBroken` message about the broken connection to $X.B$. If this is possible $X.B$ seems to have vanished from the network unexpectedly and the lane is repaired by connecting $X.B.B$ and X. Moreover, the descriptions of $X.B$'s service offers are explicitly removed from all devices by forwarding a `OfferRemove` message through the lane. If, on the other hand, X cannot reach $X.B.B$ either, the network is probably partitioned and the lane is split into two parts. In this case, X (and $X.B$ in the other half) inform their remaining lane members about the partition, which invalidate service descriptions they have stored stemming from offerers in the other half of the

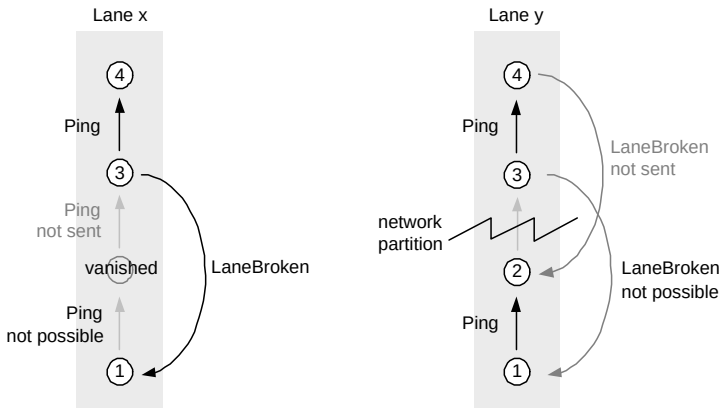


Figure 5: Algorithms for detecting broken links. On the left hand side, Node 2 has unexpectedly vanished from Lane x, which can be recognized by Node 1 and 3 because of problems with sending or receiving `Ping` messages. As the `LaneBroken` message can still be sent, the Lane can be repaired by connecting Node 1 and 3. On the right hand side, the network has been partitioned between Node 2 and 3, which can also be recognized because of `Ping` message failures. In this case, no `LaneBroken` message is received for different reasons resulting in two separated lanes.

lane. Furthermore, X adds the complete rectangle of the lower split-off part to its own rectangle resulting in the rectangle $[d^X_{x0}, d^X_{x1}] \times [d_{y0}, d^X_{y1}]$. In the same way, $X.B$'s rectangle is augmented by the complete rectangle of the upper part of the lane.

The sender of a `Ping` message detects faults in a similar way: if N cannot send its `Ping` message to $N.T$, it assumes that $N.T$ has left unintendedly. As $N.T.T$ will also recognize this problem, N waits for the appropriate `LaneBroken` message. If it arrives within a given time period, indeed, $N.T$ has vanished and the lane is repaired by connecting N and $N.T.T$, otherwise N assumes a network partition and proceeds as described above.

Note that each node in the above-mentioned algorithms is responsible for pinging *one* lane member only. In contrast to that, in CAN, each non-border node has to ensure the validity of at least four neighboring nodes, which would consume much of the available bandwidth in mobile networks.

An example for applying this algorithm can be found in Figure 5. On the left hand side, Node 2 has unexpectedly left Lane x. Node 1 recognizes this as the `Ping` message cannot be sent, Node 3 recognizes this as the `Ping` message of Node 2 is missing. Nevertheless, as the network is not partitioned, the `LaneBroken` message of Node 3 can reach Node 1, which will initiate a lane repair by connecting Node 1 and 3.

On the right hand side, the network has been partitioned between Node 2 and 3 in Lane y. The two nodes recognize this because of problems with the `Ping` message. As Node 2 does not receive a `LaneBroken` message and

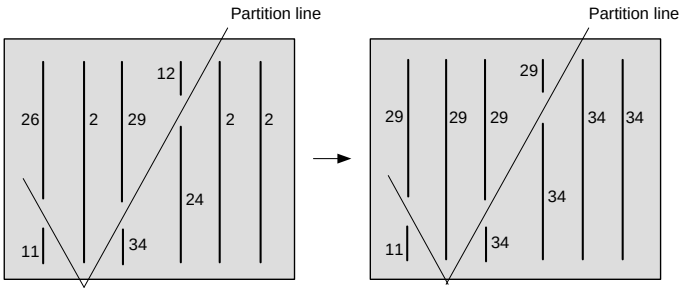


Figure 6: Propagation of partition numbers in case of network partition. Originally, every lane in the network had the partition number 2 (not shown). At a certain time, the network has been partitioned by the marked line. As three lanes are affected by this, six new partition numbers are assigned to the remaining lane parts: 26 and 11, 29 and 34, and 12 and 24 (left picture). After propagating these numbers in the own partitions, only three numbers remain leaving three uniquely denoted partitions: 11, 29 and 34 (right picture).

Node 3 cannot send such a message, both assume a network partition and start to inform their remaining lane members of the changed situation.

4.4.3 Network Partition and Reintegration

Network partition and reunion pose difficult problems for the consistency of overlay structures. In the case of lanes, network partitions can be detected with the methods from the previous section leading to two split lanes. Nevertheless, service trading can continue separately in the two parts. Furthermore, the overlay structures of the partitions can change independently.

When the radio contact between separated partitions is reestablished later, we must assure that now again services can be offered and/or found in the previously unconnected parts. Thus, a reintegration of both overlay structures is inevitable. Generally, we face two major problems: a) the reunion must be detected and b) the overlay structures of the separated parts must be combined to one new, regular overlay structure. In the following, we will analyse these problems and sketch solutions for solving them in a lanes overlay.

Detection of network reunion. To detect network reunion, it is sensible to assign different partition numbers to different partitions. Thus, when detecting network partitions by the algorithms in Section 4.4.2, each of both lane parts adds a random value taken from the set $\{1, 2, \ldots, s\}$ to its partition number resulting in a new, higher partition number[5]. After that, the two parts check, whether their originally neighboring lanes are

[5]If s is large enough, we can assume that each lane part chooses a unique value. To avoid an overflow of the strictly increasing partition numbers, the number of a partition is set back to a random number between 1 and s in times of low network changes.

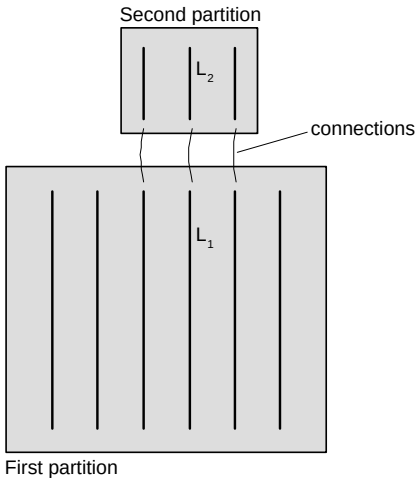


Figure 7: Schematic process of reintegrating a partition. Lane L_1 from the first partition detects the reunion of a second partition (by overhearing a `Ping` message within Lane L_2). It requests a connection between the endpoints of L_1 and L_2 to reintegrate this part. Additionally, neighboring lanes are informed and integrated in the same manner.

still reachable (via their known anycast addresses). If yes, the lane with the lower partition number overrides it with the higher one and propagates this higher value to the remaining lanes in opposite direction. On the other hand, if no previously neighbored lane can be found anymore, the lane should try to reach other neighborless lanes by broadcasting a message in the own partition. In any case, after this process, each network partition has its own unique id. Figure 6 shows an example of this idea.

To detect a network reunion, `Ping` messages are extended by the sender's partition number. Generally, we have the following rule: If Lane L_1 overhears a `Ping` message of L_2 that contains a different partition number than the own one, L_1 has detected a network reunion and reintegration steps have to be taken. As `Ping` messages are sent periodically by all nodes, network reunions are detected quickly and reliably.

Reintegration. The reintegration of two lane structures is achieved by connecting the separated lanes. To do that, L_1 prompts L_2 to establish a connection between their end nodes. In this case, the complete rectangle of Lane L_1 is horizontally halved and redistributed equally amongst the nodes of L_1 and L_2. As the old space of L_2 is overridden, L_2 loses its old anycast addresses and adopts the two addresses of L_1. Furthermore, the stored service descriptions are shared between L_1 and L_2. In case that L_1 and L_2 have already been connected earlier, this last step can be omitted by reactivating previously just invalidated service descriptions. In any case, L_2 informs its originally neighbored lanes to have them connected to its new

neighbored lanes, too. Figure 7 illustrates the process schematically.

4.5 Optimizing the Lanes Structure

A structure following the rules of being a correct lanes overlay does not necessarily yield an efficient overlay. On the one hand, the logical links constantly need to be adapted to the changing physical network conditions, on the other hand, the overall structure must fit to the usage profile.

4.5.1 Inefficient Inner Lane Connections

To improve the efficiency of the logical lane connections, the `Ping` messages for detecting broken connection can be reused. If node X receives a `Ping` message that passed a large number of hops on its way from $X.B$, it sends a `ConnectionInefficient` message back to $X.B$ to inform it about this problem. Generally, a node X does not seem to fit in a lane anymore if it receives both a `Ping` message with a high hop count from $X.B$ and a `ConnectionInefficient` message from $X.T$. In such a case, X properly logs off from the lane and logs in into another lane using Algorithm 5 and 6. Therefore, unlike in CAN, logical links are constantly adapted to the physical network topology resulting in high inner lane efficiency. [6]

4.5.2 Lane Splitting and Merging

Since nodes may enter and leave the network at any time, the length of a lane can exceed or fall below an optimal size. Generally, the proportion between lane length and number of lanes needs to be adapted to the network profile: When we examine many services searches in comparison to service announcements, a few long lanes are advantageous as for searching only a few inter lane hops are necessary. In case of many service announcements, many short lanes tend to be optimal as only a few inner lane hops are needed. Additionally, the dynamics of the network should be taken into account. In general, the number of lanes should be increased when recognizing high dynamics in order to take advantage of the unmanaged structure between the lanes. In any case, it is advantageous not to determine the optimal lane length in advance, but adapt its value constantly to the current situation in the network (or even to the situation in each lane separately). We are working on these algorithms at the moment. In one possible solution, more powerful devices continuously collect profile data, e.g., the proportion between searches and announcements, the average residence time of nodes in the lane etc. in order to increase or decrease the optimal lane length actively. In any case, we need algorithms to split oversized lanes and merge undersized ones.

[6] One of these messages can already suffice, if its hop count is extremely high.

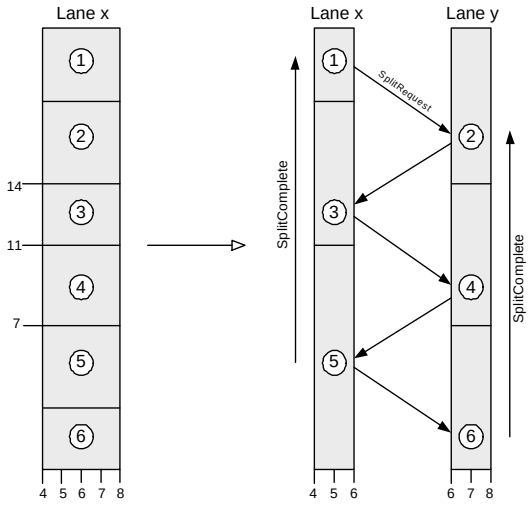


Figure 8: Splitting a lane in a zipper-like manner. Starting with Node 1, a `SplitRequest` is sent through Lane x. As result, each node alternately joins the left Lane x or the right Lane y, halves its rectangle vertically and combines it with the vertically split-off half of its predecessor. The final `SplitComplete` message helps to remove descriptions of services that are not offered in the lane anymore.

Because of the proposed optimization step presented in Section 4.5.1, we can assume that lanes are generally well adapted to the physical network topology. Therefore, when splitting a lane, we only have to assure that this ordering is preserved as well as possible. On that account, it is reasonable to separate a lane in a zipper-like manner: Nodes are traversed sequentially and alternately assigned to the two resulting lanes (see Figure 8). For that reason, nodes that become neighbored in one of the new lanes are still connected in a topology-aware manner because they have only been separated by one intermediate node in the old lane and did not stem from far distanced points in the lane. Additionally, their originally intermediate node has become a member of the other lane, so the path to that neighbored lane is topology-aware, too. Typically, the splitting starts with the topmost node in a lane, which also initiates it. This is possible, because it knows the length of its lane from previous `Ping` messages containing a counter that is set to 0 by the bottom node and increased by 1 by every lane member that is reached.

Algorithm 6 [Lane splitting]
The topmost node N wants to split its lane. To do that, a `SplitRequest` *message is forwarded through the lane. It consists of two lists,* `leftNodes` *and* `rightNodes`*, which will contain the addresses of the nodes of the two newly created lanes and will be extended by each node that is reached.*

1. *N creates a* `SplitRequest` *message, inserts its address to the* `leftNodes`

list, puts the right half of its rectangle into the message and forwards it to N.B. After that, N updates its rectangle and sets its new lower neighbor to N.B.B (known from previous `Ping` *messages).*

2. *Each node X receiving a* `SplitRequest` *checks the contained rectangle to determine in which lane it should be integrated. We assume, X shall be inserted in the right lane. Then, it removes all service offer descriptions stemming from nodes in the* `leftNodes` *list and marks all descriptions stemming from nodes in* `rightNodes` *to be pertained. After that, it updates its upper neighbor to be the last entry in the* `rightNodes` *list, inserts its own address to that list, puts the left half of its rectangle in the* `SplitRequest`*, and forwards it to X.B. Then, X combines its remaining rectangle with the one received by the* `SplitRequest` *and updates its new lower neighbor to X.B.B.*

3. *If X.B or X.B.B is empty, splitting is finished for the current lane. In this case, the remaining rectangle space is added to X and an additional bottom-up run is started by sending a* `SplitComplete` *message including a* `pertainList` *only containing X's own address to X.T.*

4. *Each node X receiving a* `SplitComplete`*, removes all service descriptions that do not stem from nodes in the* `pertainList` *and have not been marked to pertain in the top-down run. After that, only valid service descriptions from the own lane are stored on X. Finally, X inserts its address to the* `pertainList` *and forwards the* `SplitComplete` *message to X.T, if available.*

Note that by halving the rectangles vertically, each of the two new lanes retains the outer anycast address and receives a new inner one. Therefore, beside the split lane, no other lane has to be changed.

Figure 8 shows an example of this process. Consisting of 6 nodes, Lane x is too big. Therefore, a `SplitRequest` is sent through the lane, starting at Node 1. Alternately, the nodes join the left Lane x and the right Lane y. The distribution of the rectangles is demonstrated by Node 4: Originally, it owned the rectangle $[4, 8) \times [7, 11)$ and therefore reached its neighboring lanes by the two anycast addresses 3 and 8. After the splitting, Node 4 has been inserted into the right Lane y. Its rectangle is vertically halved and combined with the half of its previous predecessor (Node 3) resulting in $[6, 8) \times [7, 14)$. Thus, its neighboring lanes can now be contacted via the new anycast addresses 5 (for connecting Lane x) and 8 (as before).

The opposite of lane splitting is lane merging. It is needed if one (or two neighboring) lanes are smaller than their optimal size. As the merging algorithm resembles the splitting algorithm in many ways (merging is done in the fashion of a "closing zipper"), we will omit the details here.

5 Advantages of Lanes

The methods for service advertisement and discovery introduced above have several advantages over existing approaches. This is particularly true when taking into account the characteristics of ad hoc networks and complex, semantic service descriptions. The main reason for this is the lightweight overlay structure: Lanes. Compared to CAN's strict hypercube structure, lanes require the fulfillment of much fewer and weaker structural conditions. This makes them better suited for highly dynamic network topologies. For instance, the detection of node failures, network partitions and reintegration can be achieved with just one periodic ping message per node, whereas in the CAN structure each node needs to constantly keep track of all of its neighbors, resulting in at least four periodic ping messages for inner nodes.

Furthermore, a number of methods for optimization of lanes exist. On the one hand, they allow to adapt existing lanes to reflect the underlying network structure resulting in a more efficient communication within the lane. On the other hand, they allow for dynamic adaptation of the lane structure to the current usage profiles by splitting and merging lanes as needed. To achieve this, profile metadata about local usage and structural parameters is constantly collected and evaluated.

Compared to the approaches presented in Section 2, lanes have the following advantages: They do not need dedicated nodes for any tasks, that is, they are completely decentralized concerning structure and algorithms. Still, it is not necessary to flood the network to find or announce services. Since lanes offer a possibility for semantic service search without using semantic information to form the overlay (other than the fact that the two dimensions stand for service announcement and search, respectively), they are independent of the service description language used. On the one hand, this separation of structure and application allows for the trading of complex semantic service descriptions. On the other hand, it enables the support of arbitrary additional applications.

However, the assignment of service announcements to the inner lane structure and of service requests to inter lane communication is not arbitrary: For instance, the fixed structure within a lane allows to forego periodic refreshments of service announcements via a lease concept, since changes in the set of service offers are propagated immediately within the lane.

To summarize, lanes are the ideal compromise between weakly structured approaches, which are easily adapted to network characteristics but typically scale poorly, and highly structured approaches with optimal adaptability to user profiles at the cost of highly inefficient maintenance in dynamic network topologies.

6 Conclusions and Future Work

In this paper, we provided an overview of existing service discovery approaches in ad hoc networks. In order to overcome their drawbacks, we identified CAN as a promising basis for service discovery and introduced an application that copes with semantically rich service descriptions. The mismatch of CAN based service discovery and MANET's profile led us to ease CAN's structural restrictions by conceiving Lanes. We proposed protocols that maintain Lanes in the presence of diverse events that are typical of ad hoc networks. Lastly, we pointed out the benefits of Lanes compared to existing service discovery approaches.

In future, we intend to specify algorithms for automatically tuning network parameter (like lane lengths or time between `Ping` messages) to the profile of the network and the user. Furthermore, we are currently examining the advantages of hierarchies of lanes, which are formed by aggregating complete network regions into clusters that act as "nodes" in a higher lane structure.

Undoubtedly, it must be our prime task to gain experimental results of the proposed mechanisms. Therefore, we are currently implementing all presented protocols for testing in a MANET simulator.

Acknowledgement

The work done for this report is partially funded by the German Research Community (DGF) in context of the priority program (SPP) no. 1140 [27]. We thank Daniel Pfeifer for his valuable input while proof reading the paper.

Bibliography

[1] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proceedings of ACM SIGCOMM 2001. (2001) 161–172

[2] CORBA: Trading object service specification. (http://cgi.omg.org/docs/formal/00-06-27.pdf)

[3] Sun Microsystems: Jini. (http://www.jini.org/)

[4] Bluetooth Sig.: Bluetooth. (http://www.bluetooth.com/)

[5] Napster: Protocol specification. http://opennap.sourceforge.net/napster.txt (2000)

[6] Network Working Group: Service location protocol - RFC 2165. http://www.ietf.org/rfc/rfc2165.txt (1997)

[7] UDDI: Universal description, discovery and integration. (http://www.uddi.org/)

[8] LEAP: Lightweight extensible agent platform. (http://leap.crm-paris.com)

[9] Tarkoma, S., Laukkanen, M.: Supporting software agents on small devices. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. (2002) 565–566

[10] Internet Engineering Task Force: Simple service discovery protocol - internet draft v1.03. (http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt)

[11] Gnutelliums: Gnutella. (http://gnutella.wego.com)

[12] Waterhouse, S.: JXTA search: Distributed search for distributed networks. Sun Microsystems Whitepaper - http://search.jxta.org/JXTAsearch.pdf (2001)

[13] Defense Advanced Research Projects Agency: DARPA agents markup language - services (DAML-s). (http://www.daml.org/services/)

[14] Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of ACM ASPLOS, ACM (2000)

[15] Bakker, A., Amade, E., Ballintijn, G., Kuz, I., Verkaik, P., van der Wijk, I., van Steen, M., Tanenbaum, A.S.: The globe distribution network. (In: Proceedings of the USENIX Annual Conference) 141–152

[16] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications, ACM Press (2001) 149–160

[17] Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. In: International Workshop on Design Issues in Anonymity and Unobservability. (2001)

[18] Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (2001)

[19] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., Lilley, J.: The design and implementation of an intentional naming system. In: 17th ACM Symposium on Operating System Principles (SOSP 99). (1999)

[20] Ratsimor, O., Chakraborty, D., Tolia, S., Kushraj, D., Kunjithapatham, A., Gupta, G., Joshi, A., Finin, T.: Allia: Alliance-based service discovery for ad-hoc environments. In: ACM Mobile Commerce Workshop. (2002)

[21] Klein, M., König-Ries, B.: Multi-layer clusters in ad-hoc networks - an approach to service discovery. In: Proceedings of the First International Workshop on Peer-to-Peer Computing (Co-Located with Networking 2002), Pisa, Italy. (2002) 187–201

[22] Klein, M., König-Ries, B., Obreiter, P.: Service rings – a semantical overlay for service discovery in ad hoc networks. In: The Sixth International Workshop on Network-Based Information Systems (NBIS2003), Workshop at DEXA 2003, Prague, Czech Republic. (2003)

[23] Joseph, S.: NeuroGrid: Semantically routing queries in peer-to- peer networks. In: Proceedings of the International Workshop on Peer-to-Peer Computing (Co-Located with Networking 2002), Pisa, Italy. (2002) 202–214

[24] Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS). (2002)

[25] Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable networks. In Crowcroft, J., Hofmann, M., eds.: Proceedings of the Third International COST264 Workshop (NGC 2001). Number LNCS 2233, London, UK, Springer-Verlag (2001) 14–29

[26] Network Working Group: Reserved IPv6 subnet anycast addresses. http://www.ietf.org/rfc/rfc2165.txt (1999)

[27] Deutsche Forschungsgesellschaft (DFG): Schwerpunktprogramm 1140: "Basissoftware für selbstorganisierende Infrastrukturen für vernetzte mobile Geräte". (http://www.tm.uka.de/forschung/SPP1140/)