3 Related Work

The convergence of P2P overlay networks and mobile ad hoc networks is a quickly evolving field that has been attracting large amounts of research. This chapter will first give an overview of related work that could be used to provide the building blocks for distributed applications in mobile ad hoc networks. Following that, 5 concrete examples of related work will be presented.

3.1 Conventional Distributed Hash Tables

Distributed Hash Tables – such as [45, 50, 56, 73] – provide very efficient key-based overlay routing. Hence, when trying to build large-scale distributed applications in mobile ad hoc networks, one could simply employ the conventional Distributed Hash Tables that have been successfully used as buildings block for such applications in the Internet. Those DHTs could then provide *application-layer* indirect (i.e. key-based) routing to the respective distributed applications. However, as already stated in Section 1.1, conventional Internet-based DHTs are ill-suited for a mere deployment on top of MANETs. This is due to the follow reasons:

1. Bear in mind that, in conventional DHTs, two overlay neighbor nodes are not likely also to be physical neighbors. Therefore, overlay hops often incur unnecessarily long physical routes, which can lead to a considerable overlay stretch (see Section 2.3). In other words, the physical route traveled during an overlay key lookup can literally zigzag through the physical network (again, see Figure 1.1 and Figure 2.5). While mechanisms have been proposed to alleviate this problem (see Section 2.3), this still poses a severe problem in MANETs. Here, the packet delivery probability decreases with each additional physical hop due to factors such as packet collisions or transmission errors.

2. Due to perpetual node movement, routes in MANETs are usually quite volatile and break quickly. Therefore, the efficacy of an application-layer DHT quickly deteriorates when the physical route to carry out an overlay hop has to be (frequently re-) established by the underlying ad hoc routing protocol. For example, it is easy to imagine a situation where a key lookup requires two overlay hops, both of which have to have their physical routes discovered through broadcasting by the ad hoc routing protocol. Clearly, in that case, one would have been better off broadcasting the key lookup itself in the first place instead of triggering *two* broadcasts.

3. In order to guarantee routing convergence and consistency, DHTs have to periodically maintain their overlay routing tables. Depending on the size and structure of a DHT's routing table, the maintenance traffic can generate a significant amount of traffic. Given the limited bandwidth in MANETs,

conventional DHT maintenance can be prohibitively heavy-weight and massively interfere with other packets.

Therefore, conventional application-layer Distributed Hash Tables will not perform well on top of mobile ad hoc networks. This is further corroborated by our findings in Section 2.3.

3.2 Unstructured Peer-to-Peer Networks for MANETs

3.2.1 Flooding-Based Protocols

Early work on the convergence of peer-to-peer overlay networks and mobile ad hoc networks have dealt with the straight-forward implementation of unstructured P2P overlay for such MANETs. Those approaches combine ad hoc routing and unstructured overlay flooding (usually by using the route discovery mechanisms of the ad hoc routing protocol to also look for keys) to locate the desired object in the network. Below, two representatives of unstructured P2P overlays for file-sharing in MANETs will be presented in detail.

In [27], a P2P file sharing protocol for MANETs named ORION is proposed. ORION combines Gnutella-style [17] flooding and AODV [41] ad hoc routing to locate requested files in the network. With ORION, each node in the MANET has a local repository containing the files that the nodes is sharing. When a node now wants to locate a certain file, it issues a query message that is broadcast through the network. Whenever a node receives such a query message, it sets up the reverse route to the originator – just as AODV does with its RREQ packets – and retransmits the query message to its physical neighbors. Furthermore, each such intermediate nodes checks its local repository for any files that match the description (e.g. file name, key words, etc.) specified in the query message. If such a file / files is / are found, the node will send a response message containing the identifiers of all matching files back to the requester using the AODV-style reverse route. Each intermediate node on the response path will also update its file information cache with the file identifiers contained in the response message and the provider (i.e. the sender of the response message). After the requester has received a response, it will then send a data request for (one of) the file(s) to (one of) the provider(s) using the AODV-style routes discovered during the search. The provider will then divide the requested file into blocks and send data packets containing the various blocks of the requested file back to the requester.

In [20, 54], the Mobile Peer-to-Peer (MPP) protocol stack is proposed for file sharing in MANETs. Very similar to ORION, the integral part of MPP is a routing protocol named EDSR that combines Gnutella-style flooding and DSR [24] ad hoc routing. When a node wants to locate a desired file, it will issue a search request that is flooded throughout the MANET. Whenever a node receives such a search request, it will communicate with its application using the MPP protocol stack to see if the application can provide a matching file. Each intermediate node adds its own node address to the search request to create a DSR-style route and retransmits the search request to its physical neighbors. If the application can provide the requested file, a reply message will be send back

to the requester using the reverse path information as contained in the search request. After the requester has received a reply, it will download the desired file from the provider using HTTP on top of EDSR that routes none-search packets in the same way as DSR does.

The integration of unstructured, flooding-based peer-to-peer overlays and reactive ad hoc routing is an intuitive and simple solution for the discovery of objects in MANETs. It is a straight-forward approach as the changes and enhancements to the underlying ad hoc routing protocols are minimal since reactive protocols already have the capability of broadcasting requests and directly replying to the requester. What sort of packets (e.g. search requests, file transfer packets, etc.) are being broadcast or unicast is really of secondary importance to the ad hoc routing protocols. However, first and foremost, the obvious downside of such approaches is their poor scalability. It is clear to see that the network-wide broadcast of search requests will not scale to both growing network sizes and increasing request rates. Furthermore, since reply and file transfer messages are unicast, their reliability depends entirely on the scalability and performance of the chosen (reactive) ad hoc routing protocol.

3.2.2 Unstructured Key Lookup

An unstructured key lookup service for MANETs is proposed in [3]. With KELOP, each node assigns itself a random hash key. Similarly, each service or object is assigned a hash key in the same hash space. As usual, the node whose own hash key is numerically closest to a given key among all participating nodes is responsible for that key. The main concept of KELOP is that it does not maintain any explicit multicast structure. Instead, it solely relies on the local route cache of the given ad hoc routing protocol. When a service is advertised or requested, a message containing the service's hash key is sent to the node currently responsible for the key. For that purpose, each forwarding node simply checks its local route cache to determine the node that has the closest known hash key to the packet's key and the packet is then forwarded to that node. This process continues at each forwarding node until the packet reaches a node whose own hash key is numerically closer to the packet's key than the hash keys of all other nodes in its local route cache. In that case, the node considers itself responsible for the packet and stores the advertisement (received advertisement packet) or replies (received request packet) with a (previously stored) advertisement.

KELOP does not incur any additional maintenance traffic besides the ad hoc routing protocol's. In the case of a reactive routing protocol, there will be no or only a negligible amount of maintenance traffic. However, there are two *striking* shortcomings in KELOP. First, since KELOP nodes do not even maintain their numerical successor, there is no way in which KELOP could guarantee any deterministic routing convergence. When a node finds itself numerically closest to a packet's key with respect to the nodes in its local route cache, the probability that that node is indeed responsible for the packet's key is very low. The validity of such a decision depends entirely on the content of the node's local routing cache and, thus, upon pure chance. Thus, the likelihood that the advertisement of a node's service and a subsequent request for that service issued by a node in a

different area of the network will indeed be delivered to the same node is quite small. This effect will become even more pronounced as the network size increases.

Not only does KELOP not provide deterministic routing, the accumulated route to an eventually wrong target node can also significantly deviate from an optimal direct path. Should the nodes' local route caches also contain routes to nodes other than immediate one-hop neighbors, it is easy to imagine how KELOP could route service advertisements or discovery on a zigzag route through the network.

3.2.3 Proactive Search Routing for Mobile Peer-to-Peer Networks

In [28], a step is taken towards reducing the heavy overhead of always broadcasting search requests. The Zone-based P2P (ZP2P) protocol is based on the concept of local zones. Each node i is the center of its own zone. The zone radius is determined by a fixed hop count k so that all node that are at most k hops from node i belong to i's zone. Initially, a new node sends out an advertisement message containing information on all the data that the new node is sharing. This advertisement message will be broadcast inside its zone – i.e. within a radius of k hops. The one-hop neighbors of the new node will further reply with a message containing their own advertisements as well as those of more remote members of the zone that they have already received. This way, all nodes within a zone will know of all data that the zone members share. Whenever the content of a node changes – e.g. because it has downloaded a new file that it will now share or because it is no longer sharing a certain object – the node will broadcast an update advertisement within its zone.

When a node is interested in a certain object, it will first check its local cache to see whether any of its zone members can provide the desired object. If so, it will contact that member directly to acquire the object. However, in case the requested object is not available in the node's own zone, it will initiate a *bordercast* of the request. For this, the requester will forward the request to its border nodes, i.e. to those of its zone members that are exactly k hops away. Upon reception of the request, each border node checks its local advertisement cache to see if there is a node in its zone that can provide the requested object. If so, it will forward the request to its zone member. In order to route back a reply to the requester, each node that forwards a request temporarily maintains an AODV-style reverse route by storing for each request (as determined by the request ID) the previous hop from which it has received the request. In case a border node finds that there are no members in its zone that could provide the requested object, it again will continue the bordercast by forwarding the request to its own border nodes. This process continues until either a predefined TTL expires or the network has been searched.

By introducing the concept of local zones into the P2P search process, some of the network-wide broadcasts may become unnecessary. However, whether or not a requested file can be provided by nodes inside the requester's own zone depends entirely upon chance. Especially in larger networks, the cases where a request could be satisfied locally can be expected to be rare. Hence, the utility of local zones will evidently not scale with growing network sizes. The propagation of

requests using bordercasts can lower the overall traffic as a certain number of inner nodes might not have to forward the requests. Nonetheless, with growing network sizes, the bordercast process will quickly encounter the same problems of a regular broadcast as the number of zones that need to be contacted also increases. Furthermore, the efficacy of a bordercast depends entirely on factors such the zone radius and the node density inside the zones. In networks with low or medium node density, it is likely that the routes from the center node of a zone to its border nodes will involve most (if not all) of the inner nodes. Thus, in such networks, the bordercast will closely resemble a regular broadcast. Hence, in such networks, the performance of ZP2P can be expected to be worse than that of a regular broadcast due to the additional continuous (update) advertisement messages that need to be exchanged. Although unfortunately not explicitly addressed in [28], it is evident that nodes need to periodically re-issue their advertisements to take into consideration the effects of node mobility on zone memberships. This will further add to the additional traffic that ZP2P will generate compared to a regular broadcast application.

3.3 Structured P2P Network for MANETs

3.3.1 Chord-Based

In [9], a system called ISPRP is proposed that takes a first step towards integrating Chord's [56] structured overlay routing and ad hoc routing. ISPRP is concerned with initiating a Chord-like overlay ring among the nodes of a MANET without the need for any underlying ad hoc routing protocol. The general idea is that each node assigns itself an overlay ID. Each node further only stores the overlay IDs of its one-hop neighbor as well as the overlay IDs of its numerical successor and predecessor node in the overlay ID space and maintains a DSR-style [24] source route to both its successor and predecessor. To achieve this, a new node i selects that node j with the smallest overlay ID among its one-hop neighbors whose overlay ID is at the same time larger than the node i's own ID. In other words, node i greedily assumes that its successor node is among its one-hop neighbors. Node i the sends a *Successor Pointer Solicitation (SPS)* message to node j telling node j that node i has chosen j as its successor. An SPS message always contains the DSR-style source route from the sender to the recipient (in the case of a one-hop communication this is trivial, but this will become important later on for the resolution of inconsistencies).

Whenever a node j receives an SPS message from some node i, it will check its node cache for a possible inconsistency. If node j cannot detect any inconsistency, it will store node i as its current predecessor node along with the source route contained in the SPS message. However, in case node j does detect an inconsistency, there are two possible (even simultaneously occurring) causes. First, it could be that node j itself has been pointing to an incorrect successor node. This would be the case if node j detected a node x in the source route contained in the SPS message whose ID is smaller than node j's current successor's ID but still larger than node j's own ID. In this case, node x should become node j's successor, and node j will itself send an SPS message to node x

using the source route contained in node i's original SPS message. The second inconsistency that node j could detect is that there is already another node y pointing to node j as its successor. Since, in an ordered overlay ID ring, a node cannot have two predecessors, node j will check whether i or y should be its predecessor. W.l.o.g. suppose that node y should remain j's predecessor. In this case, node j will select from its local node cache the node z that j thinks should be i's successor instead. Node j will then send a *Successor Rewiring Solicitation (SRS)* message to inform node i about this containing the concatenated source route from node i to node z via node j so that node i can adjust its successor pointer to node z. Additionally, j will also send an SPS directly to node z on behalf of node i, again containing the concatenated source route from node i to z via j. Whenever a node receives an SPS or SRS message, the node will check for local inconsistencies and the process just described may repeat itself.

ISPRP can initiate a doubly-linked (each node stores its successor and predecessor) Chord-style overlay ID ring in a MANET without the additional need for any ad hoc routing protocol. In a network thus set up, packets could be routed based on a packet key to the node currently responsible for that key in a straight-forward manner. However, ISPRP leaves numerous essential issues unaddressed. First of all, it is clear to see that the concatenated source routes contained in the SPS and SRS messages will usually not represent shortest paths between the respective nodes. Thus, nodes will usually store source routes to their respective successors and predecessors that will deviate significantly from the optimal (i.e. shortest) paths to them. The authors in [9] also identify this problem and state "To reduce path lengths, nodes shorten their successor paths using Dijkstra's algorithm". How this is to be done with nodes only knowing their successor, predecessor, and one-hop neighbors remains completely unclear. Secondly, since nodes in a network set up with ISPRP only know their successor and predecessor (and their random one-hop neighbors), the key-based overlay routing in such a network is highly suboptimal. Obviously, a key lookup could take up to n/2 overlay hops (with n being the number of nodes in the network) in the worst case. This problem is further aggravated by the fact that ISPRP does not take physical locality into consideration so that each overlay hop – i.e. the forwarding of a packet from the current node to its successor or predecessor – can travel arbitrarily long physical routes. In other words, a key lookup could literally zigzag through the network. This is *still* further aggravated by the already mentioned problem that nodes will usually not even know the *shortest* paths to their successor and predecessor. Arguably the most severe insufficiency of ISPRP, however, is that it does not address network dynamics. In MANETs, routes between nodes break frequently due to node movement. Unfortunately, ISPRP does not address how to detect and repair such route break-ups. Furthermore, ISPRP also does not consider node failures. No detection and repair mechanisms are proposed to handle situations where a node fails, which essentially results in a broken overlay ID ring.

Based upon ISPRP, a *Scalable Source Routing* (SSR) protocol is proposed in [15]. After bootstrapping the network, each nodes has acquired the source route to its numerical success node. By adding the routes contained in the packets received at a node to the local route cache, nodes aim at gathering O(log N) source routes

to nodes whose address space distances increase exponentially (resembling a Chord-like routing structure). As with the original ISPRP, SSR has a couple of significant drawbacks. First of all, as described above, the accumulated source routes acquired during the successor updates can be highly suboptimal (compared to the shortest paths between a node and its numerical successor). To alleviate this problem, nodes use the source routes in their routing caches to prune unnecessarily long source routes – e.g. routes containing cycles or a shorter sub-path to one of the nodes in the source route is known (short cut). However, the effectiveness of this source route pruning entirely depends upon the available cache entries and there are no guarantees as to how well the source routes in the system can be pruned. The second – and arguably the most severe – drawback of SSR is that physical locality is not explicitly taken into consideration. As described above, a key lookup can thus literally zigzag through the physical network. This is further aggravated the fact that – as just discussed – the source routes of the individual overlay hops are quite likely to be suboptimal – in other words: not only can a key lookup be expected to zigzag through the network, but the individual legs of the zigzag course will also quite likely be unnecessarily long. Clearly, this is especially unfavorable in mobile ad hoc networks. Unfortunately, the authors do not present any results in [15] as to how SSR would perform in mobile networks and/or networks with churn.

Most recently, another DHT-like routing scheme for MANETs has been proposed [4]. *Virtual Ring Routing* (VRR) works in a manner quite similar to SRR. With VRR, each node maintains an AODV-style route (for a destination, the next physical hop is stored in the local routing table) to each of its r virtual neighbor nodes, i.e. to the $r/2$ nodes whose virtual identifiers are numerically closest to the node's own virtual identifier but smaller as well as to the $r/2$ nodes whose virtual identifiers are numerically closest to the node's own virtual identifier but larger.

Routing with VRR is a straight-forward task. When a node wants to send a packet with a given key, it selects from its routing table the entry that is numerically closest to the packet's key. The packet is then sent off using the next hop information from the routing table entry. When a node receives a packet, it again checks its routing table to find the numerically closest destination node and forwards the packets towards that node. This process continues until the packet reaches the node whose virtual identifier is numerically closest to the packet's key ,and who is, thus, responsible for the packet.

When a new node A joins the network, it asks one of its physical neighbors B that is already member of the network to send a setup request to the node currently closest to node A's virtual identifier. That node Y will then add node A to its routing table and respond with a setup message that contains node A's r virtual neighbors. Upon reception of the setup message from node Y, node A will add node Y to its own (still empty) routing table. Next, node A will contact its other r-1 virtual neighbors to setup the physical routes to them.

Similar to SSR, VRR has to a couple of significant drawbacks. First of all, since nodes setup physical paths to new virtual neighbors through existing physical neighbors, VRR is likely to suffer from the same suboptimal physical routes as

SSR does. Also, there is no correlation between physical locality of nodes and their virtual identifiers. Thus, here too, a key lookup can be expected to zigzag through the physical network. Again, this even further aggravated by suboptimal physical routes. These drawbacks become visible in the results presented in [4]. For network sizes of ≥ 150 nodes, VRR's packet delivery ratio deteriorates drastically, which raises doubts about the scalability of VRR. Unlike SSR, however, the authors analyze their protocol for mobile networks and present a thorough description on how to handle link failures [4].

3.3.2 Pastry-Based

Arguably the most closely related approach to MADPastry is Ekta [42]. Ekta integrates Pastry key-based routing and DSR ad hoc routing at the network layer to provide a DHT substrate explicitly designed for mobile ad hoc networks.

Each Ekta node maintains a standard Pastry leaf set and routing table. However, instead of the conventional <overlay ID, IP address> entries, Ekta leaf set and routing table contain the overlay ID and a DSR-style source route from the current node to the respective entry node. When a new node i wants to join an Ekta network, it assigns itself a random Pastry overlay ID and asks its bootstrap node to start a lookup towards that key. Once the join request is received by the node j that has so far been responsible for node i's overlay ID, it will reply with a message containing its leaf set. Unlike standard Pastry, this reply message will not contain any routing table information in order to keep the traffic overhead as low as possible. Node j will also broadcast a message to all the leaf set members to inform them about the new arrival.

Key-based routing in Ekta works analogously to Pastry's routing. When a node I wants to send a packet with a key k, it looks up the destination node j of the next overlay hop in its Pastry routing table or leaf set that has a longer common prefix with key k than the current node does or, if no such entry exists, that is at least numerically closer to k – as stipulated by standard Pasty routing. Once the destination node for the next overlay hop has been determined, the source route stored along with that entry is used to execute the overlay hop. This process is continued at each overlay hop destination until the packet eventually arrives at the node that responsible for the packet's key.

If node i chooses for the next overlay hop a destination node j for which the physical does not exist (any longer), Ekta differentiates between two cases. If the node was chosen from the leaf set, then, by definition of the Pastry leaf set, there will not be any alternative leaf set entry and Ekta initiates a DSR route discovery for the path from node i to node j. If, on the other hand, node j was chosen from the routing table, one does not necessarily have to discover a route to node j itself. Instead, it suffices to discover the route to *any* node that would also fit into the same routing table slot that node j was taken from. Therefore, in such a situation, Ekta starts a prefix-based route discovery to find the route to any such node.

In order to reduce the maintenance overhead as much as possible, Ekta tries to update its routing tables and its source routes using the information gained from overheard packets. Therefore, instead of engaging in the costly Pastry routing

table maintenance procedure, whenever an Ekta nodes receives or overhears a packet, it will use the overlay ID and source route contained in the packets to replace less optimal routing table entries (i.e. those further away) and to refresh source routes. This way, Ekta can significantly reduce the maintenance overhead compared to an application-layer DHT that merely "sits" on top of an ad hoc routing protocol. Although not mentioned in [42], it is evident that Ekta nodes will have to engage in explicit leaf set maintenance to prevent the overlay ID ring from breaking.

Ekta combines Pastry key-based routing with DSR ad hoc routing at the network layer to provide an efficient DHT substrate for mobile ad hoc networks. The simulation results in [42] indicate that such an integrated DHT substrate achieves a significantly better performance than an application-layer DHT does in MANETs. However, Ekta does not explicitly consider what we believe is one of the essential issues in MANETs: physical locality. In Ekta, there is no correlation between overlay proximity and physical proximity. Instead, Ekta nodes rely on the chance overhearing of packets to optimize their routing tables with regard to physical locality and to, thus, reduce the overlay stretch in the network. This shortcoming is further aggravated by the fact that Ekta nodes need to perpetually communicate with their leaf set members in order to keep the overlay ID ring from breaking. Since, in Ekta, overlay neighbors, will usually not be physically close to each other, leaf set maintenance messages can travel arbitrarily long routes. This, in turn, increases the likelihood that these essential maintenance messages might be lost. Furthermore, especially in larger networks, the key lookup process in Ekta can be expected to zigzag through the network as the overlay routing approaches the eventual target node since the longer the shared prefix between the current node and the packet key, the less and less alternative destinations will exist (also see [5] for a thorough description of this effect) for the next overlay hop. Therefore, the scalability of Ekta can be expected to be limited in larger MANETs. Unfortunately, the authors of [42] only provide simulation results for relatively small networks sizes of ≤ 150 nodes.

Moreover, as has been described above, when an Ekta node i does not know the physical route to a selected node j from the routing table, it will start a prefix-based route discovery to *any* node that would also fit into the same routing table slot. Unfortunately, it is not explained how such a prefix-based route discovery should be carried out. Clearly, a simple broadcast request for any matching node to reply would practically resemble a regular network-wide DSR route request for the path from node i to node j since the requesting node i would have no control over the propagation of the prefix-based route discovery. Another conceivable solution would be to implement such a prefix-based route discovery as an expanding ring search. However, since there is no correlation between physical and overlay proximity in Ekta, it remains doubtful whether Ekta nodes could benefit from such an approach. Whether or not an Ekta node finds a matching node within a certain hop radius would depend entire upon chance so that nodes would frequently have to expand their search to the entire network.

3.4 Service Discovery in MANETs

Another set of approaches that could be used as building blocks for distributed applications in mobile ad hoc networks are service discovery protocols for MANETs. A considerable amount of research has been dedicated to the problem of service discovery in MANETs. It is beyond the scope of this thesis to present every proposed solution in this field in detail. Therefore, we will focus on the various concepts and present representative approaches.

3.4.1 Broadcast-Based Service Discovery

The easiest way to discover services or objects in MANETs is to simply broadcast a search request network-wide. Whenever a node receives such a search request, it checks locally whether it can provide the desired service or object, and, if so, it will send a response back to the requester. Such a straight-forward solution is described in [12]. Here, request are always broadcast and nodes reply using which ever ad hoc routing protocol is currently being used in the MANET. The requester would then also use that ad hoc routing protocol to access the requested service.

Other approaches try to directly integrate the service advertising and discovery with the ad hoc routing protocol. One such approach is presented in [38]. Nodes periodically broadcast their service advertisements within a certain hop radius. Since there is no separate ad hoc routing protocol any more, each node that receives such an advertisement stores along with it the previous hop from which it has received the advertisement. This way, an AODV-style route is constructed back to the service provider. Search request are similarly broadcast. Again, forwarding nodes store an AODV-style reverse path back to the requester. When a node receives a request, it checks locally whether its cache contains a matching advertisement and, if so, responds with that advertisement. While this is a simple solution, the authors unfortunately leave essential issues completely unaddressed. For example, since there is no separate routing protocol employed, how does the requester know the route to service provider that is included in the advertisement? Would it contact the provider via the responding node that it has received the advertisement from? Clearly, this would be *highly* suboptimal. Another unaddressed issue is what should be done in case the route to the provider that is constructed during the advertisement broadcast breaks.

A more convincing solution is presented in [16]. Here, the AODV routing protocol is augmented with an additional message type for service discovery. It is routed (i.e. broadcast) the same way as a regular route discovery but intermediate nodes check whether they can provide the requested service instead of a route. If so, they respond with service reply that is routed similarly to a route reply. For all other purposes – such as service access – regular AODV is used. A similar approach is taken in [33] where the service discovery is integrated with OLSR. Here, OLSR's MultiPoint Relay forwarding is used to propagate service advertisements and discoveries throughout the network. Efforts have been made to standardize the general concept of integrating a broadcast-based service discovery with reactive ad hoc routing protocols in [13].

Broadcast-based service discovery protocols for MANETs are very similar in spirit to unstructured peer-to-peer networks for MANETs, as described in Section 3.2. In fact, it is not easy to distinguish any differences at all with the exception that the unstructured peer-to-peer systems presented in Section 3.2 seem to focus on the special purpose of file discovery and the subsequent transfer of such files instead of generic services. Therefore, broadcast-based service discovery protocols exhibit the same obvious scalability problems as unstructured peer-to-peer systems do in MANETs.

3.4.2 Multicast-Based Service Discovery

Other service discovery protocols for MANETs try to limit the overhead generated by broadcast-based protocols by introducing multicast trees along which services are advertised and requested. With Konark [22], every node maintains a topic-based registry tree to store known service descriptors. Nodes periodically advertise their services to the network. For service discovery, a request is propagated within a multicast group and each receiving node checks its registry tree for a matching service description with which to reply. Unfortunately, several essential questions remain unanswered. For example, it remains completely unclear how a multicast group is formed and maintained. The provided example merely states that all participating nodes join the same "locally scoped" multicast group. This would essentially mean that a request is always broadcast within the entire network of participating nodes. Furthermore, it is not explained how the categories at each level of the registry tree are defined. This callow impression is further increased by the failure of the authors to provide any experimental results.

A more elaborate and convincing approach is taken in [29]. Here, the system constructs a virtual backbone of backbone nodes (Virtual Access Points – VAP) that are always 2 or 3 hops from each other. Local hello messages are used to establish routes between neighboring VAPs and to periodically maintain the virtual backbone. Regular nodes associate themselves with the closest VAP and register their services with it. For service discovery (and network-wide advertisements) VAPs construct multicast trees on the virtual backbone (i.e. among themselves) on demand. When a node wants to discover a service, it sends a request to its VAP. The general idea is now that each VAP is the root of its own virtual backbone multicast tree. Therefore, for each multicast source (i.e. multicast tree root), each VAP maintains a list containing those of its neighboring VAPs to which the VAP should forward a respective multicast message (as determined by the multicast source). As already said, such multicast trees are constructed on demand. Therefore, a VAP's forwarding list for each multicast source initially contains all of its neighbor VAPs. Generally speaking, when a VAP receives a multicast message, it will prune the previous VAP from which it has just received the message from its forwarding list for the given multicast source as that VAP has obviously already seen the message. Furthermore, if a VAP i receives a duplicate multicast message from a neighboring VAP j to which it had not forwarded the message, it will send a prune message to that neighbor VAP instructing j to explicitly remove i from its forwarding list for the given multicast source. Hence, the initial propagation of a message from a particular VAP source node will resemble a regular broadcast

among all VAP nodes. Subsequent message propagations starting at that VAP source, however, will be able to use the thus constructed multicast structure. While this multicasting can reduce the discovery (and/or network-wide advertisement) overhead in stable or slow moving networks, its efficacy can be expected to deteriorate rapidly in more dynamic networks. In such networks, the virtual backbone topology will be quite volatile as VAPs step down and other regular nodes step up to become VAPs in order to retain the backbone distance constraints (2 or 3 hops between VAPs). Evidently, whenever a VAP learns about a new VAP neighbor (or the failure of an old one), it will basically have to reinitialize its forwarding lists as the respective multicast trees could have been severely altered. Therefore, in more dynamic networks, service discovery can be expected to closely resemble a regular broadcast among the virtual backbone nodes.

Multicast-based service discovery protocols for mobile ad hoc networks can reduce the advertisement and discovery overhead compared to broadcast-based protocols. The efficacy of multicast-based protocols is especially pronounced in stable and slow-moving networks where the multicast structure can reduce the amount of traffic. However, in more dynamic networks, the multicast structure will become quite volatile. The maintenance of multicast structures will become increasingly difficult and will constitute a significant overhead. In such dynamic networks, multicast-based protocols will often resemble regular broadcast-based approaches.

3.4.3 Cluster-Based Service Discovery

Other service discovery protocols work with the notion of node clusters. In [52], local directory nodes are selected so that each node has at least one such directory node within a radius of H hops. Regular nodes register their services with the directory node(s) in their H-hop cluster. Directory nodes periodically advertise their presence to their H-hop clusters. To reduce the traffic overhead, these directory advertisements are propagated inside their respective clusters by repeated 2-hop bordercasts until they have been distributed over H hops. Regular can also use these advertisements to update their route information to their directory nodes. If a regular node has not heard from any directory node over a certain period of time, it will initiate a directory election request that, again, will propagated over H hops using repeated 2-hop bordercasts. Nodes willing to become directory nodes (depending on their available capacities), will respond to the initiator. The initiator will then select the most appropriate (e.g. as given by its available capacity) node as its local directory and register its services with that new directory node. Furthermore, directory nodes periodically exchange their contents among each others. For this purpose, directory nodes summarize their contents using a Bloom filter representation of the service advertisements they have stored in their directories. Directory nodes now periodically advertise their Bloom filters over a radius of $2*H$ hops to other directory nodes. These advertisements can also be used by directory nodes to maintain routes to their neighbor directory nodes. For service discovery, nodes send a service request to their local directory node. Upon reception of such a service request, the local directory node checks its directory for any matching service advertisements. If found, it will reply to the requesting node. If no matching service advertisement

is available, the local directory node will start a global service request. For this purpose, the directory node probes the Bloom filters of its known other directory nodes to select those directory nodes that are likely to have a matching service discovery in their respective directories and forwards the request to them. This process is continued at each receiving directory node until the request reaches a directory node that can provide the requested service advertisement.

The concept of exchanging content descriptions among directory nodes can reduce the service discovery overhead in relatively small and stable networks. In larger and more dynamic networks, on the other hand, the efficacy of such an approach will deteriorate quickly. It will become less and less likely that a forwarding directory node happens to know of other directory nodes that are likely to have the requested service advertisements. Therefore, in such larger and more dynamic networks, the global service discovery will closely resemble a broadcast among the directory nodes with the known scalability issues of such systems. This seems to be corroborated by the small network size (only 90 nodes) that was chosen for the provided experimental results. Furthermore, the selection process for the forwarding of service requests remains somewhat unclear. Throughout the paper, it is explained that directory nodes probe the Bloom filters of other known directory nodes to select those that are likely to have a matching service advertisement in their directories. However, the brief outline of that forwarding decision algorithm seems to only consider the remaining battery lifetime of the known directory nodes and their distances. Moreover, another critical issue that is not addressed in the paper is that of the value of the cluster radius H. The authors merely vaguely state that the value is "dependent on the node density". Clearly, the value of H will have a significant impact on the overall performance of the system. For example, if it is chosen too large, the cluster diameters will also become too large and the routes from regular nodes to their respective directory nodes that were established during the propagation of the directory advertisements will frequently break. Depending on the underlying ad hoc routing protocol, costly route discoveries would have to be performed. Similarly, the distances between directory nodes will become larger and larger so that those inter-directory routes will also frequently break, which will equally decrease the performance of global service discoveries and incur additional route discoveries. If H is chosen too small, on the other hand, the number of directory nodes will increase significantly. Thus, during global service discovery, the likelihood that a directory node happens to know of other directory nodes that are likely to have a matching service advertisement will further decrease. Hence, global service discovery will involve a large number of nodes and the propagation of requests among those nodes will likely resemble a regular broadcast.

Another cluster-based service discovery protocol is presented in [68]. Here, the network is divided into a fixed number of M clusters, each of which is defined by a cluster head. This fixed number of cluster, M, is assumed to be known to all participating nodes. To select the cluster heads, all nodes initially broadcast their capacity to the network. Once, all nodes thus know the capacities of all other nodes, the M nodes with the highest capacities assume the roles of cluster heads and flood the network with their node IDs and respective cluster IDs. This way, all cluster heads can store the node IDs of all other clusters. Each node now

associates itself with the closest cluster head and registers all objects that it provides with that cluster head. For the purpose of object discovery, each object is further mapped to exactly one cluster head – the so-called *replica keeper* (RK). To determine the RK of a particular object, a globally known hash function is used that maps an object ID to one of the *M* cluster IDs. Therefore, whenever a cluster head receives an object registration from one of its own cluster heads, it hashes the object ID to the cluster ID of its RK. The cluster head then looks up the node ID of the respective cluster head (i.e. the replica keeper) and also sends a registration message to the cluster head. When a node now requests a certain object, it sends a request message to its cluster head. The cluster head will first check if it has a matching object registration, and, if so, will reply to the requesting node. If no matching object registration is found, the cluster head will hash the object ID to acquire the cluster ID of its replica keeper (i.e. the cluster head of that cluster) and forward the request to that cluster head. The replica keeper will then respond to the requesting node with a list of cluster heads that possess a matching object registration. In order to keep the replica keeper up-to-date, whenever a node changes its cluster membership (because it finds itself now closer to another cluster head), it will unregister all its objects with its former cluster head, who will in turn unregister with the respective replica keepers if no other nodes in its cluster can provide the respective objects. Analogously, the node will then register all its objects with its new cluster head, who will then register with all necessary replica keepers.

The system presented in [68] can provide reliable and deterministic object discovery in relatively small, stable and slow-moving networks. However, in larger and especially more dynamic networks, the systems exhibits a number of critical shortcomings. First of all, it assumes fixed cluster heads. Therefore, it does not address the question of what should happen if a cluster head fails. In that case, the entire cluster would be cut of from the network until all cluster members have selected a new cluster head and registered their objects. Perhaps the most severe shortcoming is that the location of an object is directly coupled with the cluster membership of its providers. Therefore, whenever a node changes its cluster membership, its former cluster head might need to send unregister messages to numerous replica keepers – depending on how many of the node's objects are also provided by other nodes in the former cluster – and, similarly, the new cluster head might need to send register messages to numerous replica keepers – depending on how many of the node's objects are already provided by other nodes in the new cluster. This maintenance traffic can be held relatively small only in slow-moving networks where cluster membership changes occur only occasionally and where, overall, relatively few objects are provided by relatively many nodes (i.e. the object is provided by numerous nodes in the same cluster). However, in more dynamic networks as well as in networks where nodes exhibit a high percentage of unique objects, this system will generate enormous amounts of maintenance traffic. In larger networks, the performance will further suffer from arbitrarily long routes between cluster heads as the scalability here will depend entirely on the capabilities of the underlying ad hoc routing protocol to deal with long and quickly breaking routes. Another shortcoming is the fixed number of clusters. This implies that, as the number of nodes and/or objects increases, the few cluster heads will have to

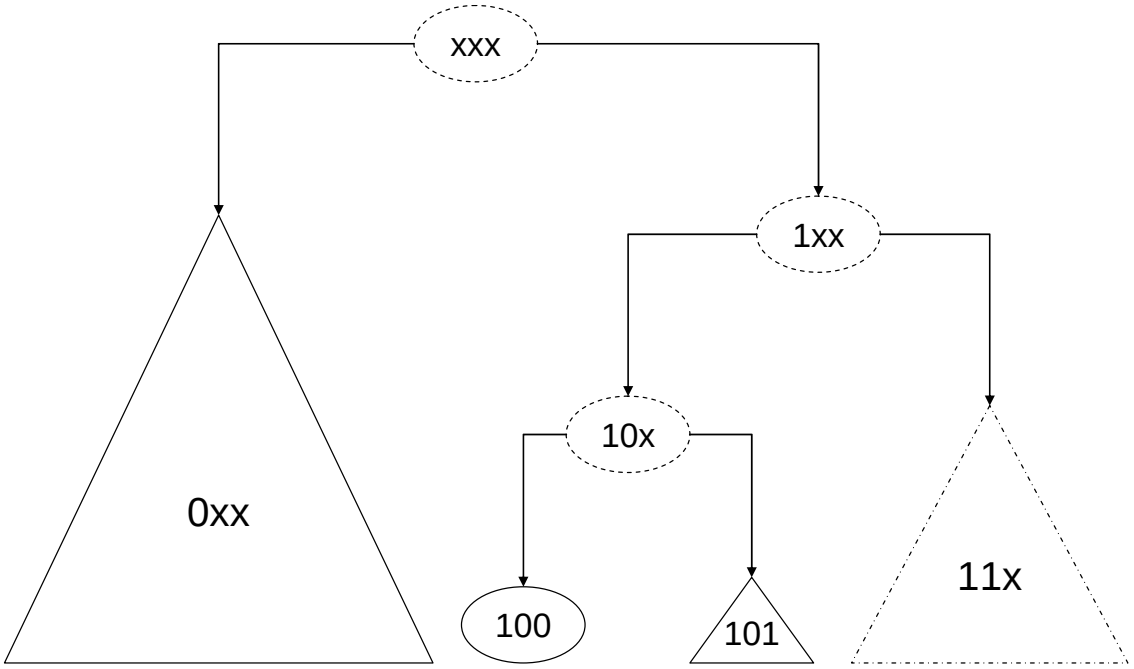

Figure 3.1 Node address tree with the *l* level-*k* siblings of node address 100. [14]

handle huge amounts of registration and discovery traffic. Therefore, the cluster heads represent bottlenecks for the scalability of the systems both in terms of the network size and the number of objects.

3.4.4 Hierarchical Service Discovery

In [14], nodes dynamically assign themselves hierarchical addresses. Such node addresses consist of *l* bits. The address space is now conceptually organized as binary tree with *l+1* levels. The leaves of that address tree represent individual nodes whereas each inner tree node defines a sub-tree containing addresses with a common prefix. Figure 3.1 (provided in [14]) shows an example of an address tree for 3-bit node addresses. To guarantee routing convergence, the system introduces a *Prefix Sub-graph Constraint* that stipulates that, for each sub-tree, all nodes (i.e. all leaves reachable within the sub-tree) in that sub-tree must form a connected sub-graph in the physical network. In other words, all nodes whose addresses are stored as leaves of a given sub-tree must be reachable among each other. For routing purposes, the system introduces the notion of a *level-k sibling*. A level-*k* sibling of a given node address is the sub-tree that exactly contains the node addresses that share a prefix of exactly *(l-k-1)* bits with the given node address. It is important to bear in mind that, due to the binary nature of the node addresses, each level-*k* sub-tree consists of exactly two level-*(k-1)* sub-trees. Hence, each node address has exactly one level-*k* sibling and, thus, a total of exactly *l* siblings. Again, Figure 3.1 shows the siblings of node address 100: its level-0 sibling is node address 101, its level-1 sibling is the sub-tree 11x, and its level-2 sibling is sub-tree 0xx.

The routing table of each node has exactly one entry for each level-*k* sibling containing the route information to any node in the respective level-*k* sibling. Thus, each routing table consists of *l* entries. When a node now wants to route a packet to a given address, the node first determines in which of its sibling trees the given address lies. It then simply uses its routing table entry for the

appropriate sibling and routes the packet to the node stored in its routing table. Due to the dynamic address allocation in the system, a node will often not be aware of the current address of its target node. For the required address resolution, the system uses a DHT-like lookup. For this purpose, each node hashes its (static) node ID into the address space. The node whose address is closest (closeness in the system is defined by the XOR value of the hash key and a node address) to that hash key will store the node's current address. Analogously, when a node A needs to resolve the address of a node B, it will compute node B's hash ID and its request will be routed to the node whose address is currently closest to the hash key.

The DHT-like name resolution in [14] could easily be used as a general-purpose service discovery mechanism as well. Services or objects would simply be hashed into the address space and, analogous to the name resolution, the node whose address is closest to the service hash key would become its temporary directory node. Due to the deterministic hierarchical routing used, the service discovery overhead will be significantly smaller compared to broadcast- or multicast-based protocols. Such a service discovery will be especially efficient in smaller and relatively stable networks. However, in more dynamic networks, a lot of address reorganizations can be expected. As nodes move around incessantly, there will be frequent violations of the strict prefix sub-graph constraint. Thus, nodes will frequently have to reassign themselves new addresses in order to satisfy that constraint. However, whenever a node acquires a new address, it will have to handover all the service advertisements that it was responsible for under its old address to the node that is now responsible for the respective service hash keys. Equally, it will have to acquire the service advertisements that it is now responsible for from the current service directory nodes. Since the prefix sub-graph constraint imposes quite a strict relationship between the address tree and the node addresses, such address changes can be expected to occur frequently in more dynamic networks, and, thus, to incur a significant amount of handover traffic. Furthermore, the routing process in the system can deviate decidedly from a direct path between the source and the destination node. This is because packets are routed form high-level sub-trees to lower-level sub-trees until they eventually reach their destination. The accumulated physical path during this forwarding will often be markedly longer than a direct path between source and destination. This will become more problematic, the larger the network becomes.

3.4.5 Geographic Service Discovery

A special class of service discovery protocols constitute the geographic-based protocols (e.g. [32, 67, 47, 57, 55, 59]). These protocols use the geographic position of the nodes (e.g. using GPS) and geographic routing to provide efficient service discovery. GHT [47], for example, uses the intuitive concept of geographic hashing. Each object is hashed into the geographic area of the network and the node who currently closest to an object's hash coordinates becomes its directory node. When a node now wants to discover the location of an object, it simply routes a request to the object's hash coordinates using the geographic ad hoc routing protocol GPSR [25]. Analogously, service providers register their services using GPSR. Other protocols such as [32, 67] divide the network area into region and distribute multiple directories for each node according to those regions.

Geographic protocols can provide efficient service discovery in MANETs. Their main advantage is that, due to the availability of the nodes' geographic positions, routes between nodes will likely follow a direct physical path. Furthermore, route maintenance becomes practically unnecessary – with geographic routing it suffices that each node merely knows its one-hop neighbors. Moreover, since with geographic hashing the overlay topology is directly mapped onto the physical topology, such geographic-based protocol will not suffer from any significant overlay stretch.

All these protocols require the availability of geographic positions – usually by means of GPS. However, additional hardware such as GPS receivers will present an additional burden on the nodes' energy resources, which are usually limited in MANETs. Furthermore, such hardware is currently usually not present on the majority of devices that form MANETs – even if the majority of hardware had such GPS receivers, they still would require are relatively unobscured line-of-sight and would not work indoors. Therefore, the general applicability of geographic-based service discovery protocols will be limited. Since the focus of this thesis lies on structured peer-to-peer services for MANETs *without* any such hardware prerequisites, geographic-based service discovery protocols will not presented here in more detail. A more extensive evaluation of geographic location services is provided in [11].

3.4.6 Geographic Service Discovery Without Location Information

In [44], a system is presented that tries to implement the concepts of geographic service discovery without the need for any explicit location inform such as GPS. The idea here is to establish virtual coordinates that so that a GPSR-style routing can be implemented. To bootstrap the network, the presence of a designated bootstrap beacon node is assumed. That beacon node broadcasts its beacon throughout the network. Nodes that are farthest away (in terms of hops) from the beacon node among all their two-hop neighbors consider themselves *perimeter nodes*. The perimeter nodes then send a broadcast message to the network so that all perimeter nodes can determine their distances to all other perimeter nodes – their perimeter vectors. Following that, perimeter nodes broadcast their perimeter vectors so that the perimeter nodes can use a triangulation algorithm to compute their coordinates. The virtual coordinates of perimeter nodes remain fixed, all other nodes periodically run an iterative relaxation algorithm to (re-)compute their virtual coordinates. At each iteration, a node computes its new x-coordinate as the sum of its one-hop neighbors' x-coordinates divided by the number of its one-hop neighbors. The new y-coordinate is computed analogously. The beacon node periodically broadcasts its beacon so that node can reassess whether they are still or have become perimeter nodes.

Based on these virtual coordinates, the system uses a GPSR-style [25] greedy geographic routing to deliver a packet based on its target coordinates to the node currently closest to the given target coordinates. For service discovery and address resolution, a GHT-like [47] DHT is used. Objects are simply hashed into the virtual coordinate space and are stored at the nodes currently closest to their respective hash coordinates.

The reliability of the system presented in [44] depends on the accuracy of the virtual coordinates. Since, for the routing of packets, a node only considers the virtual coordinates of its one-hop neighbors and then greedily forwards the packet to the neighbor closest to the target coordinates, it is essential that the virtual topology (as defined by the virtual coordinates) closely resemble the actual, physical topology. Otherwise, the probability of correct packet deliveries will decrease quickly. The presented iterative virtual coordinate computation will work well in networks with low node mobility. This impression is substantiated by the very slow average node velocity of 0.32 m/s that is used in the experiments presented in the paper. In more dynamic networks, however, nodes will have to recompute their virtual coordinates at ever shorter intervals, which would obviously generate an ever larger overhead (neighboring nodes need to exchange their coordinates for the subsequent iterations). Furthermore, the accuracy of the virtual coordinates depends on repeated iterations of the computation. With constantly moving nodes, it could well happen that nodes change their positions too quickly for the iterative computation to stabilize.

3.5 Summary

As seen, a number of quite different approaches exist that could potentially be used as building blocks for large-scale distributed network applications in MANETs. These presented approaches have often been designed for quite varying application scenarios. For example, some have been designed with a concrete application such as file-sharing in mind while other have explicitly been designed as general-purpose building blocks. Similarly, different approaches also try to optimize different network aspects such as the discovery overhead or the maintenance complexity.

The varying characteristics of the presented approaches sometimes make it difficult to compare them directly against each other. Therefore, Table 3.1 intends to broadly assess the different approaches according to the general parameters scalability, topological adaptability, maintenance complexity, general applicability and hardware prerequisites. More specifically, these general parameters are defined as follows:

Scalability. This is a measure of how well a given approach is expected to scale to large network sizes. One of the central questions is whether the employed discovery mechanisms will only work adequately in small networks because they would generate too much overhead in larger networks, or whether they could also be used efficiently in larger networks (Excellent/++ → Very poor/--).

Topological Adaptability. This measure tries to estimate how well a given approach can be expected to cope with topological changes in the physical network – for example induced by node mobility. The central issue here is whether the data structure(s) used for service/object discovery can adapt adequately to topological changes (Excellent/++ → Very poor/--).

Maintenance Complexity. This is a measure of how much maintenance overhead will be generated in order to keep the data structure(s) of the respective approaches up-to-date. Complex maintenance procedures will generate significant traffic, which will clearly influence the overall scalability as well (Very low/++ → Very high/--).

General Applicability. General applicability refers to the range of applications for which a given approach could be used as a building block. For example, some approaches are solely designed for broadcast-based file-sharing, while others can be used for name resolution but lack a generic key-to-node mapping functionality so that they cannot be used efficiently for service discovery, etc (Excellent/++ → Very low/--).

Hardware Prerequisites. This parameter indicates what assumptions concerning the hardware, that it runs on, a given approach makes. For example, a question is whether a given approach could be employed on regular devices or whether the presence of specialized hardware such as a GPS receiver is required (None/0 → many/--).

The general assessment parameters described above are weighed using the following symbols in the table below:

++ : Excellent, very low

\+ : Good, low

0 : N/a, medium, none

\- : Poor, high, some

-- : Very poor, very high, many

Table 3.1 Assessment of related approaches.

	Scalability	Topological Adaptability	Maintenance Complexity	General Applic-ability	Hardware Prerequisites
Conventional DHTs	--	--	--	++	0
Broadcast-based P2P / Service Discovery	--	++	++	--	0
Unstructured Key Lookup (KELOP)	-	0	+	--	0
Zone-based P2P	--	+	+	-	0
ISPRP / SSR	--	-	+	++	0
VRR	--	+	+	++	0
Ekta	0	+	0	++	0
Multicast-based Service Discovery	0	-	-	0	0
Cluster-based Service Discovery	0	0	+	0	0
Hierarchical Service Discovery	0	-	0	++	0
Geographic Service Discovery without Location Information	+	-	-	++	0
Geographic Service Discovery	++	++	0	++	--