6 Application I: Peer-to-Peer Based Name Service for MANETs

MADPastry provides efficient key-based routing in mobile ad hoc networks. However, as already pointed out, MADPastry is not an application by itself. Instead, it is a routing protocol that delivers a packet based on a key to the node currently responsible for packet's key in the network. Therefore, to demonstrate MADPastry's practical usability, it was next used to build a dynamic and decentralized name service for MANETs [70].

6.1 Introduction

The invisible omnipresence of the Domain Name System (DNS) [35, 36] in the Internet shields one of the most fundamental challenges from network applications and users: How to bind a resource, for example a file or a service, to a specific network address. Network resources are usually identified by some URI, e.g. "http://some_node.net/some_resource". It is essential for a network application to resolve a given URI to the concrete network address of the node where the desired resource actually resides.

However, in MANETs, centralized names services are usually not available. Therefore, MADPastry is used to build MAPNaS [70] – a decentralized name service for MANETs. In MAPNaS, a resource (e.g. a file, a service, etc) is identified by a unique resource key that is mapped into the logical MADPastry ID space. Due to the lack of a fixed network topology in MANETs, there are no dedicated resource directory servers. Instead, true to the P2P paradigm, every node functions both as a resource host (of its own files, services, etc.) and as a resource directory for certain remote resources. As determined by MADPastry, every node keeps track of the network addresses of those resources whose resource keys it is responsible for. Since mobile devices often have limited hardware and storage capabilities, the design goal of MAPNaS is to keep the architecture as simple as possible. This is demonstrated by Figure 6.1. Nodes store the resource descriptors (the resource key along with the specific network address of the resource) they are responsible for in their local MAPNaS repository. Furthermore, every node advertises its own resources that it is willing to share through MAPNaS.

6.2 Resource Advertisement

When a node A in a MADPastry network wants to make a local resource (e.g. a service, a file, etc.) available to other nodes in the network, it needs to assign a hash key to that resource, e.g. by hashing the resource's URI. Using that key,

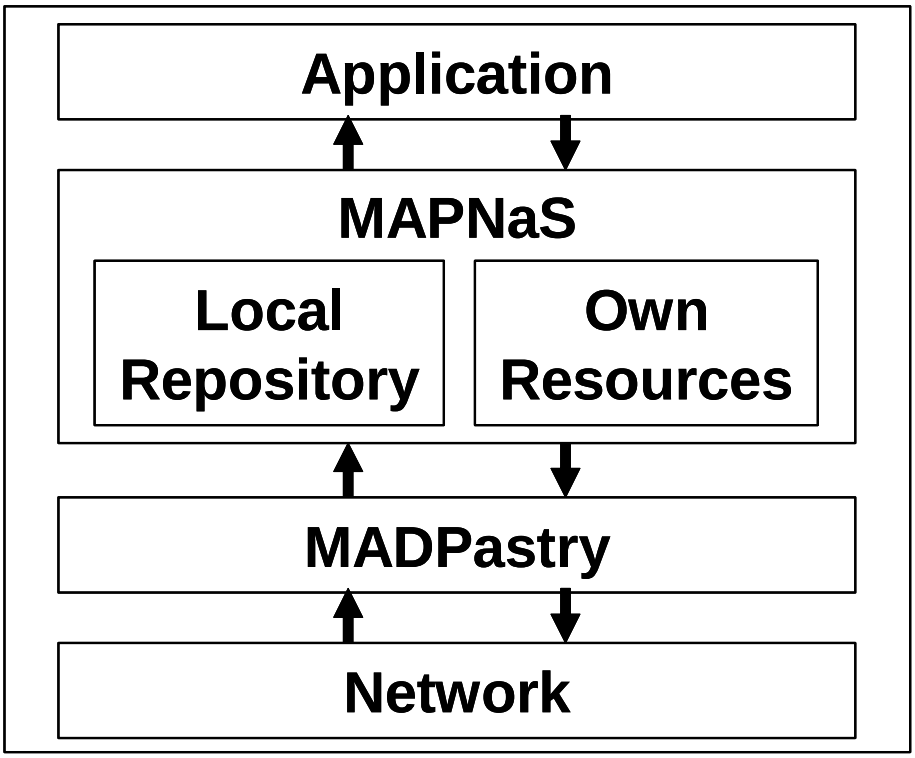

Figure 6.1 The MAPNaS architecture.

node A will then construct a resource descriptor consisting of the resource key and the physical network address (e.g. IP address) of the resource provider (in this case node A's address). Using MADPastry, the descriptor is routed to the node currently responsible for the resource key. That recipient node will then store the resource descriptor in its local repository.

Figure 6.2 shows an example of a resource advertisement. Node 17, whose current overlay ID is `75A1FFE2`, wants to advertise its resource with the URI "http://mynode.net/my_resource". Hashing that URI yields the hash key `B7E9A578`. Node 17 now constructs a resource descriptor containing the resource key and the network address of the host: `{B7E9A578, 17}`. As described in Section 4.4, this advertisement packet will then be routed to the responsible node using MADPastry. At node 17, the closest entry in its MADPastry routing table is node 4 with overlay ID `B207D11F`. Node 17, thus, sends the packet to node 4. This first overlay hop (as indicated by dotted arrow) takes 5 physical hops (as indicated by the solid black arrows) to be completed and delivers the packet from the source cluster already to the target cluster (as indicated by the two shaded regions). Node 4 will then consult its MADPastry routing table to determine the next node to forward the advertisement packet to – in this case node 35 with overlay ID `B7E1C101`. This second overlay hop consists of two physical hops. For the final overlay hop, node 35 consults its MADPastry leaf set to forward the packet to node 79 (overlay ID `B7E9A014`) who is responsible for the resource key. Upon reception of this advertisement, node 79 will store the resource descriptor `{B7E9A578, 17}` in its local repository.

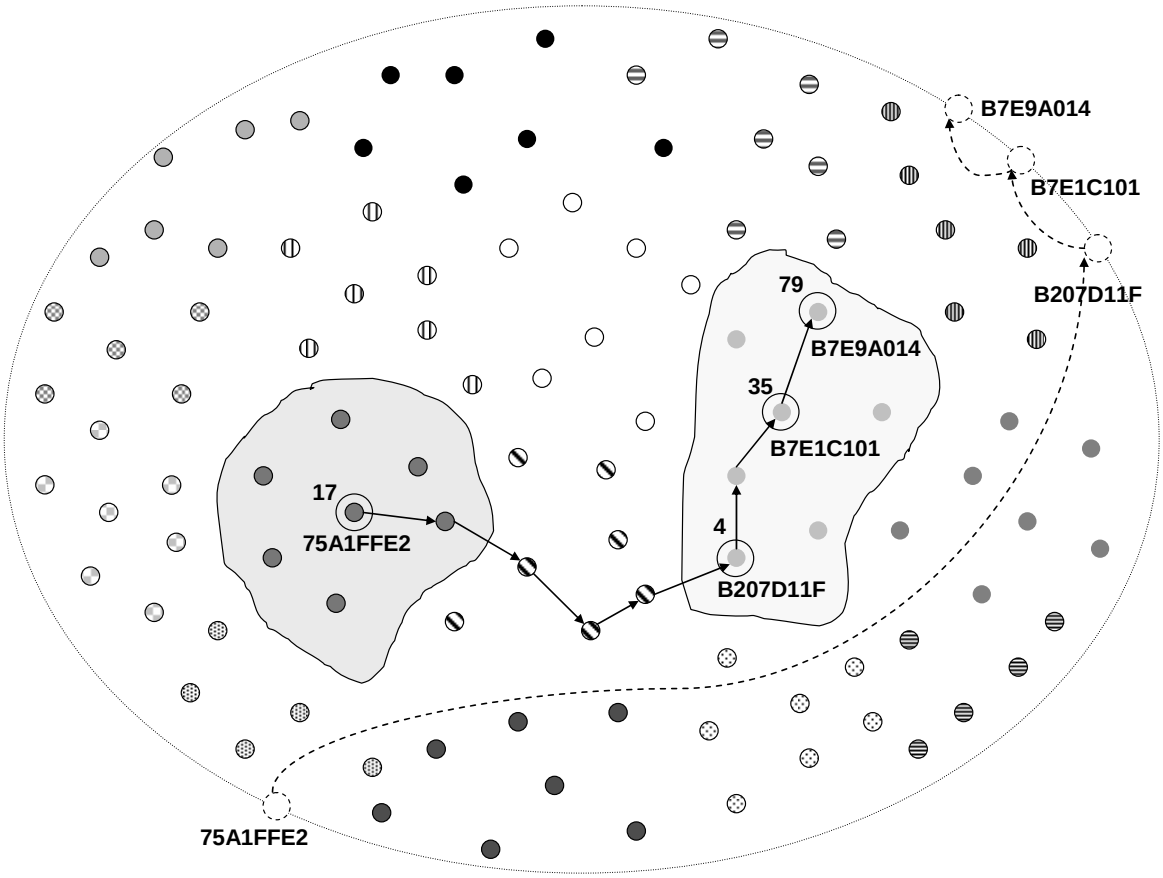

Figure 6.2 Indirect routing of a MAPNaS resource advertisement using MADPastry.

6.3 Resource Discovery

Resource discovery with MAPNaS works analogously to the resource advertisement process. When a node needs to resolve a resource's URI (e.g. "http://somenode.net/some_service"), it will simply hash the URI of the resource and send a lookup request to the node currently responsible for that hash key. This request is routed using MADPastry in the exact same indirect manner as described above for the resource advertisements. The eventual destination node will check its local repository and send back the matching resource descriptor (or multiple descriptors in case several nodes are hosting the same resource).

Figure 6.3 illustrates the resource discovery process with MAPNaS. Following up the example from the previous section, suppose now node 63 (overlay ID A101D11F) is interested in the resource "http://mynode.net/my_resource" that is provided by node 17. Unfortunately, node 63 has no idea which nodes provide the desired resource. Therefore, node 63 hashes the URI of the resource, which yields the hash key `B7E9A578` as it did for node 17 for its advertisement. Next, node 63 simply sends a request for a matching resource descriptor towards the hash key using MADPastry. Thus, in the first overlay hop, the request will be delivered to node 35 with overlay ID `B7E1C101`. Node 35 will then forward the request to its leaf set member node 79 with overlay ID B7E9A014 who, as before with the resource advertisement, is responsible for the given hash key. Upon reception of the request, node 79 will check its local repository and send a response

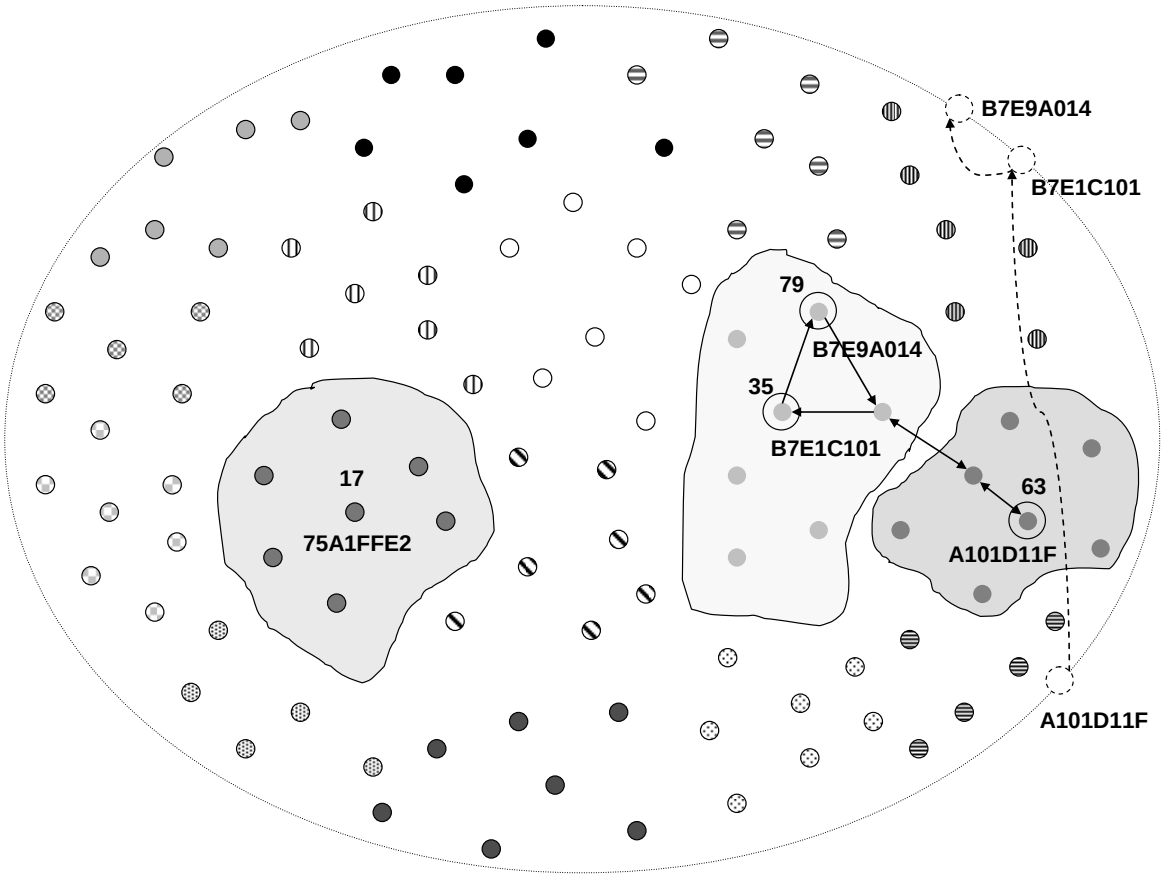

Figure 6.3 MAPNaS resource discovery using MADPastry routing.

containing the resource descriptor {B7E9A578, 17}, that it had previously received from the provider node 17, back to the requester, node 63. At this point it shall be remarked that for such a resource response, MAPNaS also uses indirect, key-based routing instead of a direct AODV-style unicast from the resource directory to the requester. The reasons for this will be presented in detail in Chapter 7.

6.4 Local Replications

For the scalability and feasibility of a MANET, it is essential to restrict network traffic to local regions as much as possible [19, 31]. Therefore, MAPNaS makes use of MADPastry's clusters to store local replications of resource descriptors.

When a node intends to advertise a resource, it will now insert the resource descriptor under two different keys. The first key is the regular hash key (of the resource's URI, etc.) and the resource descriptor is inserted into the network as described above. To obtain the second key under which the resource descriptor is stored, the regular resource key is altered to make sure the descriptor will be stored in the resource host's own MADPastry cluster. For this purpose, the resource key's prefix is replaced with the host's own cluster prefix. In a MADPastry network with 16 landmark keys (i.e. 16 prefix-based clusters), node 17 from the previous example would store the descriptor for its resource "http://mynode.net/my_resource" first under its regular hash key **B**7E9A578

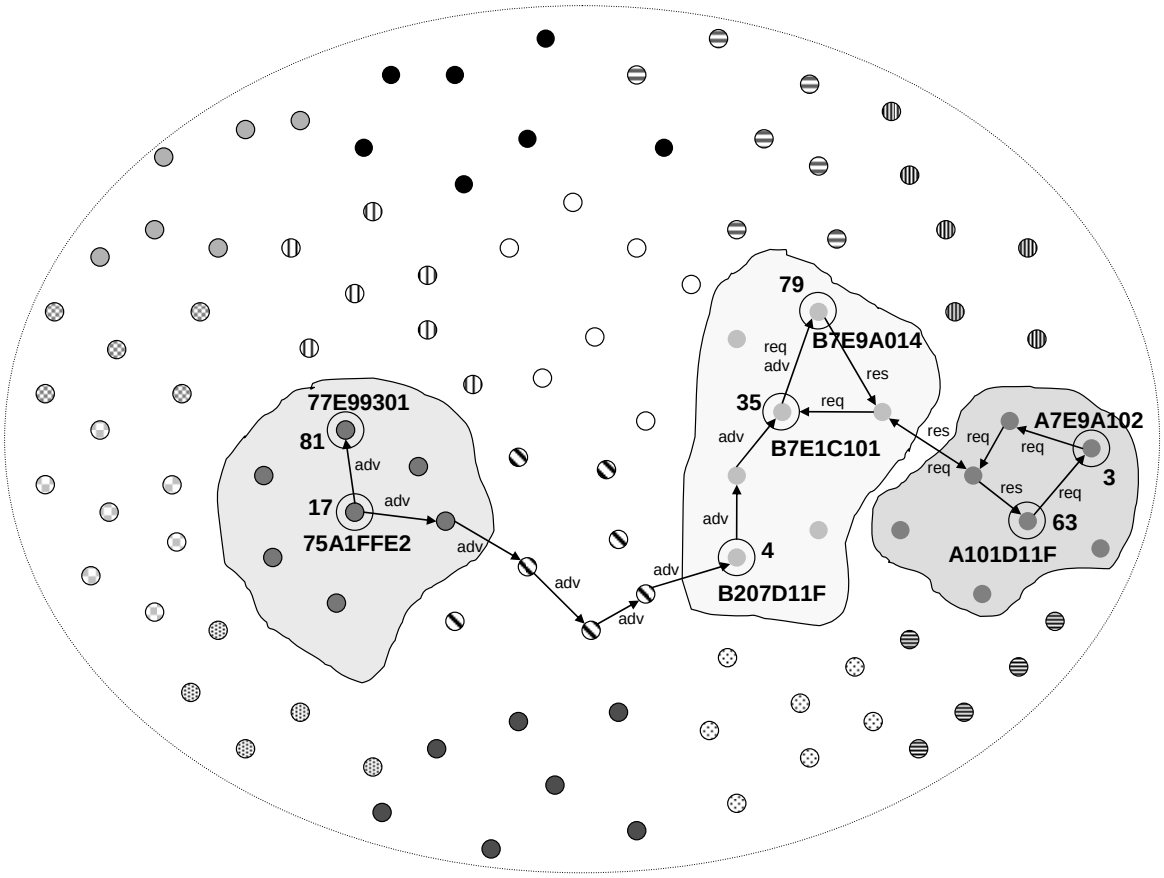

Figure 6.4 Resource advertisement and discovery with local replications.

somewhere in the network. Additionally, it would also insert the resource under the local key **7**7E9A578 into its own MADPastry cluster.

As described in Section 4.2, MADPastry clusters are made up of nodes that share a common overlay ID prefix so that they are close to each other in the overlay ID space. With Random Landmarking, these overlay neighbors are also likely to be close to one another in the physical network. Hence, intra-cluster communication can be expected to travel only short physical paths. Therefore, when a node needs to lookup the address of a certain resource, it will generate the regular resource hash key (by hashing the resource's URI) as described above. Before engaging in a potentially cross-network indirect routing process to find the corresponding resource descriptor, the node will first replace the descriptor key's prefix with its own cluster prefix. To restrict the lookup process – if possible – to nodes in its physical vicinity, the lookup request is then first routed to the appropriate local cluster member to see whether a matching descriptor can already be found in the local cluster. This might, for example, be the case with popular files or standard services that are hosted by multiple nodes. Only if this local lookup provides no (appropriate) answer, will the request be forwarded as in a regular network-wide lookup process.

This process is illustrated in Figure 6.4. When node 17 wants to share its resource "http://mynode.net/my_resource", it will generate its regular hash key B7E9A578. As before, node 17 will then send an advertisement towards that key that will eventually be delivered to node 79 whose overlay ID is closest to the resource's hash key. Additionally, node 17 will also store its advertisement locally

in its own cluster. For this purpose, node 17 generates the local key for the resource by replacing the prefix of the hash key with its own overlay ID prefix, which yields the key **7**7E9A578. It will then use MADPastry to route the extra advertisement to that local key, which will be delivered to node 81 with overlay ID 77E99301 who is responsible for the local key. Analogously, when node 63 is interested in the resource "http://mynode.net/my_resource", it too will generate the resource's regular hash key B7E9A578. However, before issuing a global resource discovery request, node 63 will first check its own local cluster for a matching resource descriptor. For this purpose, node 63 produces the local key **A**7E9A578 and routes a request using MADPastry to the node responsible for that key – in this example node 3 with overlay ID A7E9A102. Node 3 will then check its local repository for a replication of a matching resource descriptor and reply to node 63 with a found descriptor or, otherwise, node 3 will forward the request under its global hash key so that will be eventually delivered to node 79 as described before.

6.5 Handovers and Caching

When a MADPastry node moves from one cluster to another, it will eventually join the new cluster by assigning itself a new overlay ID that shares a common prefix with its cluster members. Therefore, when a MADPastry node running MAPNaS changes its cluster membership, it needs to pass the resource descriptors that are in its local repository to its old "left" and "right" leaf set members as those two nodes will now be numerically closest to the corresponding resource descriptor keys. Furthermore, when the rejoin process under its new overlay ID is done, it needs to acquire from its new "left" and "right" leaf set members those resource descriptors for whose keys it has now become responsible.

Since a handover packet could be lost – e.g. due to collision, etc. – a node can potentially end up having some resource descriptors in its local repository that it is actually not responsible for (any longer). To take care of such incidences, each node periodically checks its local repository for such descriptors and hands them over, if need be, to the best candidates as proposed by its Pastry leaf set or routing table.

Furthermore, in MANETs, a node typically overhears a good number of packets that are not destined for it. Exploiting this virtually cost-free (in terms of network traffic) extra information, a MAPNaS node caches the information of all advertisement, handover, and lookup response packets that it overhears. Since the storage capabilities of mobile nodes are usually scarce and since resource descriptor can become stale, these cache descriptors are tagged with a simple timestamp (e.g. 30s) after which they expire. Of course, cached descriptors are not handed over in case of a cluster membership change. When an intermediate node on the physical path of an overlay hop is to forward a lookup request, it first checks its own local cache to see whether it can already satisfy the request.

6.6 Experimental Results

To evaluate the performance of MAPNaS, a MAPNaS reference application was implemented running on top of a MADPastry routing agent in ns-2. The fundamental question to be answered in MANETs when dealing with elaborate approaches such as MAPNaS running on top of MADPastry, is whether the effort of maintaining the data structures is really worthwhile. In the case of MAPNaS, the question is whether there is really anything to be gained from going through the process of advertising resources, handing over resource descriptors, maintaining MADPastry's routing tables, etc. Or, would it be absolutely sufficient if nodes did not advertise their resources at all and if resource discovery requests were, instead, simply broadcast through the network? For this purpose, a second reference application was implemented as well where nodes do not publicly advertise their own resources, and where resource discovery requests are simply broadcast (already forwarded requests will not be forwarded a second time). Every receiving node checks its own resources and if there is a match, it sends back a direct response using AODV.

Similar to the previous chapter, the following metrics are analyzed:

Success Rate. This is the percentage of random requests that eventually deliver a response containing the correct resource descriptor back to the originator node. In other words, this is the round-trip (request + response) success rate of all resource discoveries.

Packet Overhead. This is the total number of packets that are forwarded during the entire simulation. This count is increased whenever a node forwards a packet to the next physical hop – in other words, whenever the MAC layer of a node is being passed a packet down from an upper layer. Hence, it includes all application and router packets such as AODV route requests and replies, MADPastry maintenance packets, and MAPNaS resource advertisements, requests and replies.

Overall Traffic. Since many different packet types (e.g. AODV route requests, MADPastry packets, resource advertisements, handover packets, etc.) of various packet lengths are transmitted during a simulation run, not merely the total packet count is analyzed. Additionally, the total network traffic in Kbytes that is created during the simulated hour is also considered. Whenever a node forwards a packet, this figure is increased by the packet size. Again, this figure includes *all* routing and application level packet types (AODV, MADPastry and MAPNaS packets).

As in the chapter before, all simulations that were carried out modeled wireless networks of 250 nodes over the course of one (simulated) hour. Nodes are always moving around according to the random way point model with 0s pause time (constant movement) and at a steady speed of 1.4 m/s – a quick walking speed. For data transmission, nodes are, again, using the 802.11 communication standard with a transmission range of 250m. The node density in the investigated networks is always 100 nodes/km². All MAPNaS nodes periodically

Table 6.1 Simulation parameters and values – basic results.

Simulation Parameter	Value range
Simulation duration	3600s (simulated)
Network size	250
Network density	100 nodes/km²
Node mobility	1.4m/s (constant, 0s pause time)
Random request interval (per node)	10s
Resources shared per node	5 (1250 total)
Local lookups first	no

send out requests for a randomly picked resource. Again, a 32-bit overlay ID space is assumed with hexadecimal overlay IDs. In other words, each overlay ID consists of 8 hexadecimals digits. Please refer to Appendix 9.2 for MADPastry's system parameters and the values used throughout this section.

6.6.1 Basic Results

In a first set of simulations, MAPNaS on top of MADPastry is compared against the simple broadcast approach with AODV. In both applications, each node issues a request for a random resource descriptor every 10 seconds. Furthermore, each node shares 5 resources. To measure the basic results of both systems, nodes do not try to lookup the resources in their own cluster first by fixing the respective resource keys, but, instead, they always lookup the resources' regular hash keys

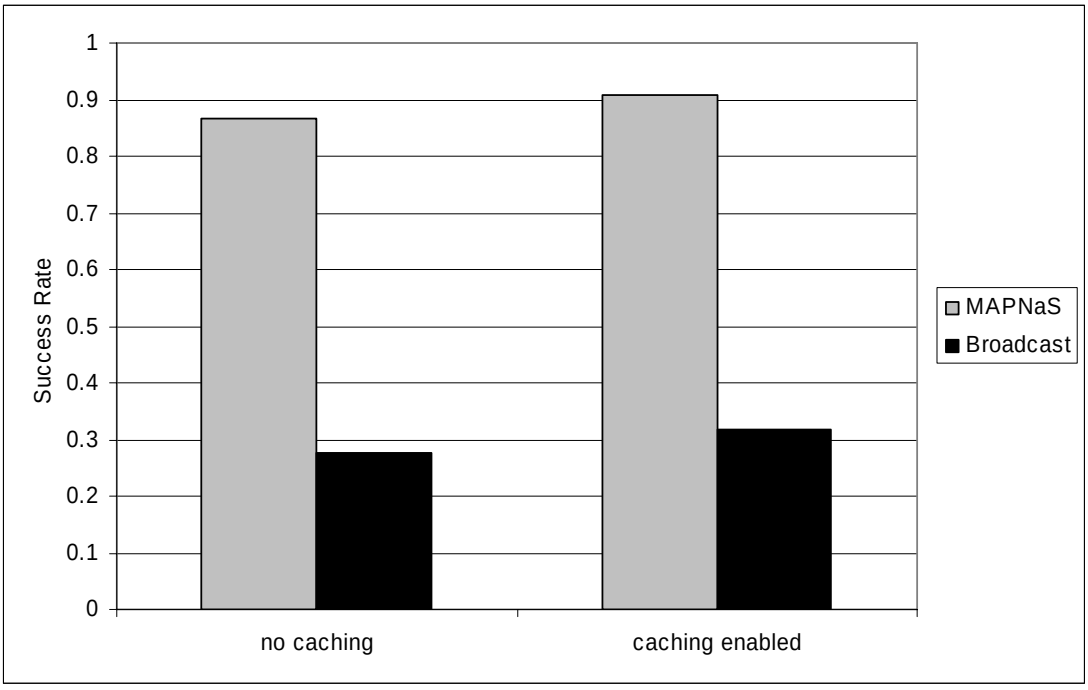

Figure 6.5 Success rates - basic results.

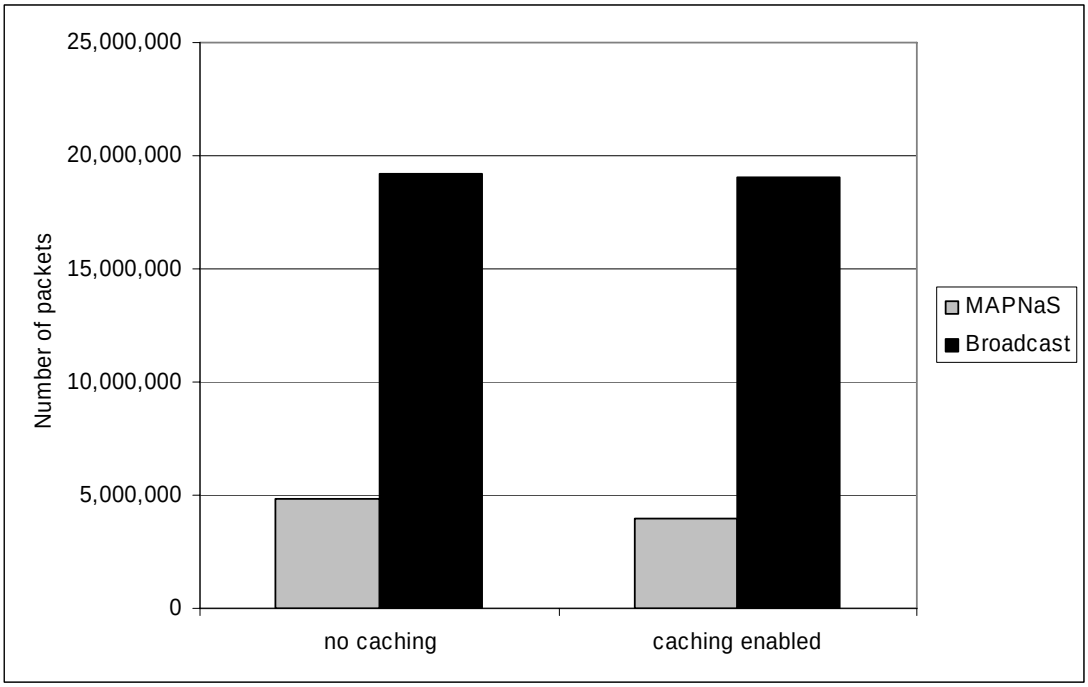

Figure 6.6 Total number of packets sent.

right away (although nodes still publish local replications of their resources when they change cluster membership). Table 6.1 provides an overview of the simulation parameters and their respective values.

Figure 6.5 compares the success rates of MAPNaS against the broadcast approach. Without the caching of overheard packets, MAPNaS on top of MADPastry achieves a drastically higher success rate than the broadcast approach does (87% vs. 28%). When nodes cache the content of packets they overhear, a modified version of the broadcast application is used as a network

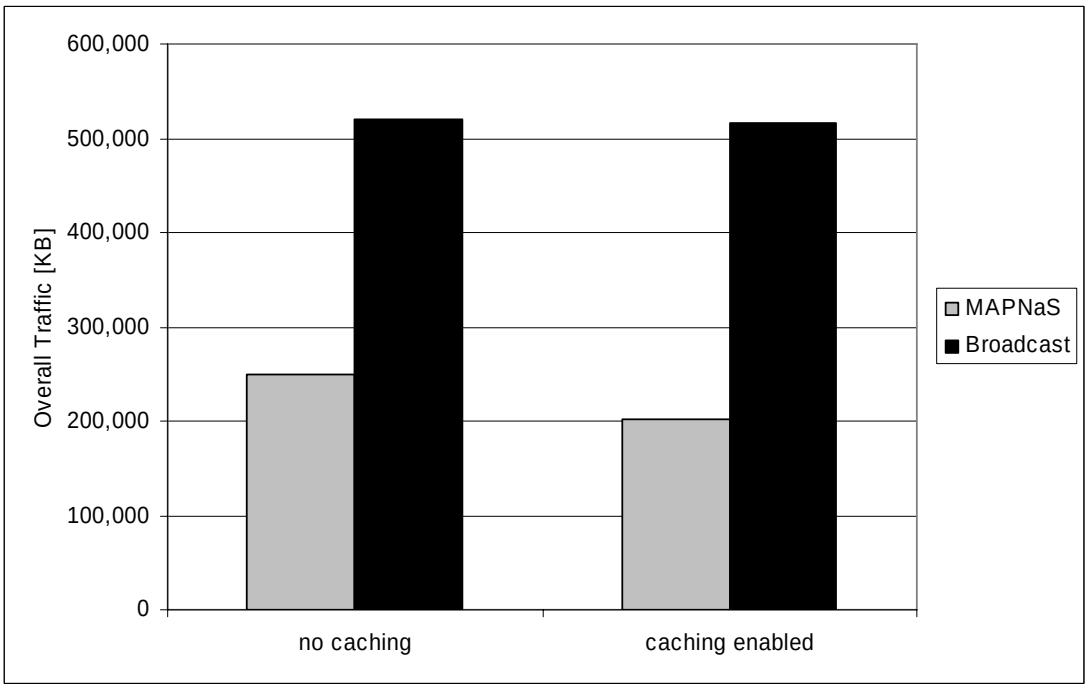

Figure 6.7 Overall traffic.

Table 6.2 Simulation parameters and values – local replications.

Simulation Parameter	Value range
Simulation duration	3600s (simulated)
Network size	250
Network density	100 nodes/km²
Node mobility	1.4m/s (constant, 0s pause time)
Random request interval (per node)	10s
Resources shared per node	5 (1250 total)
Local lookups first	yes
Traffic pattern	Random requests, 80% requests for local resources

wide broadcast could hardly benefit from local caches (even if a nearby node could already satisfy an originator's request, it could not prevent other nodes from still forwarding the broadcast request). Therefore, a simple form of expanding ring search is used instead. The broadcast application will first limit the propagation of its requests to a TTL of 3. Only if no cached entry could be found in that local region, will the request be broadcast throughout the network. Without updates, objects remain in the cache for 30s. As can be seen, both applications can further increase their success rate in combination with caching, but the difference is still huge (91% vs. 32%).

There are two main reasons for MAPNaS's much better performance. First of all, in a 250-node network, the broadcast application produces significantly more traffic (between 2-3 times more) than MAPNaS over MADPastry does – both in terms of the number of forwarded packets as well as in terms of the traffic generated overall – as Figure 6.6 and Figure 6.7 demonstrate. Because of this higher traffic, there are clearly more packet losses with the broadcast application due to factors such as collisions and interference. Thus request and responses are often dropped before they reach the appropriate destination. Secondly, the overall traffic of the broadcast application is entirely made up of broadcast requests, AODV packets, and response messages, all of which usually affect the entire network. With MAPNaS on the other hand, a sizable portion of the overall traffic stems from MADPastry and is thus often restricted to certain clusters.

6.6.2 Local Replications

In the next set of simulations, the benefits of local replications are examined. For this purpose, three different traffic patterns are considered. First of all, as a reference line, the scenario from the previous section is presented again where nodes request random resources using their respective regular hash keys right away and do not try to lookup the resources' replications in their local clusters first ("random – no local tries"). In the second traffic pattern ("random – try

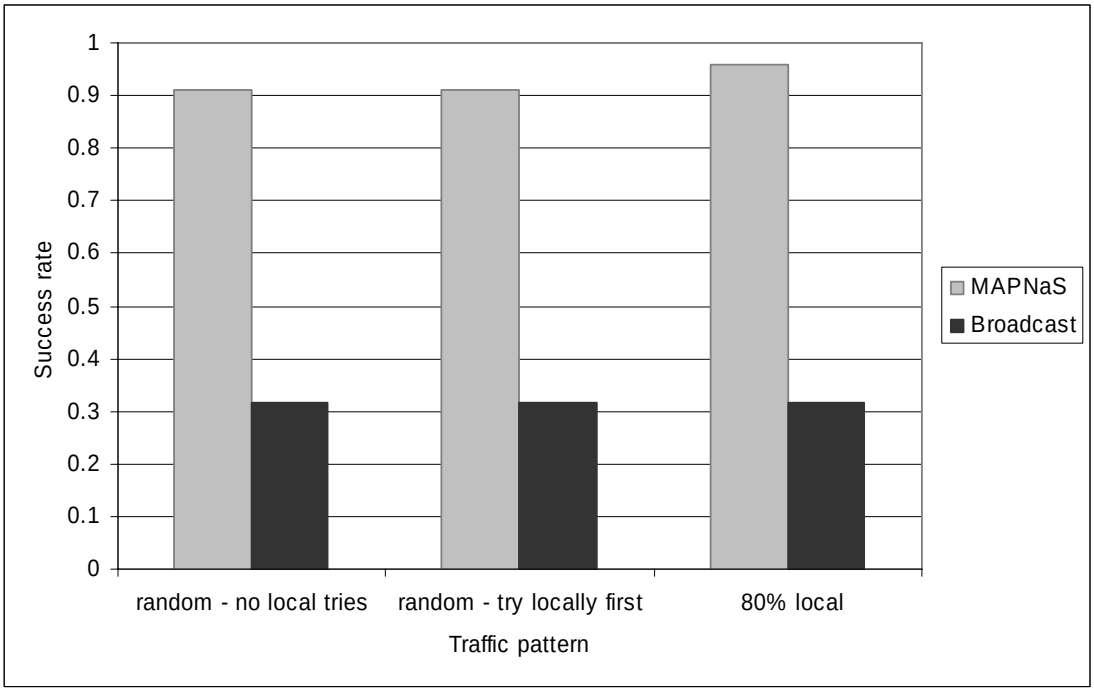

Figure 6.8 Success rate vs. traffic pattern.

locally first"), nodes still request random resources. This time, however, a node always tries to first lookup a possible replication of the desired resource in its local cluster. In the third traffic pattern ("80% local"), nodes will turn out to request resources that are hosted by nodes from their own cluster (without actively being aware of this) in 80% of the cases. Table 6.2 provides an overview of the simulation parameters and their respective values.

As there are no overlay clusters in the broadcast application that would allow for a deterministic way of inserting local replicas and since nodes do not advertise their resources there, it is hard to compare the performance of MAPNaS with local replications against the broadcast application in a fair and meaningful manner. For the sake of simplicity and to provide a reference line, the expanding ring search from the previous section is used as an approximation.

Figure 6.8 shows the success rates that MAPNaS and the broadcast-based approach achieve in the various traffic patterns. As can be seen, local replications should be favored in situations where nodes are likely to request resources that are hosted in their vicinity. When nodes display a behavior where they request resources that are hosted by nodes in their own cluster in 80% of the cases, the success rate is increased to over 95%. This is because local requests (i.e. requests that are sent to nodes in the same cluster) are very likely to travel shorter physical paths and are thus more likely to be successfully delivered. On the other hand, in scenarios where nodes request resources that are hosted by arbitrary nodes, the (often unsuccessful) initial local lookups do not help improve the success rate. This is because by first issuing a local request, that is unlikely to be satisfied and thus likely needs to be redirected to the global directory node, the accumulated path length of a resource request increases, which, in turn, adversely affects the probability of a successful delivery (and, thus, cancels out the positive effects of the occasional local hits). Please note that, even in this

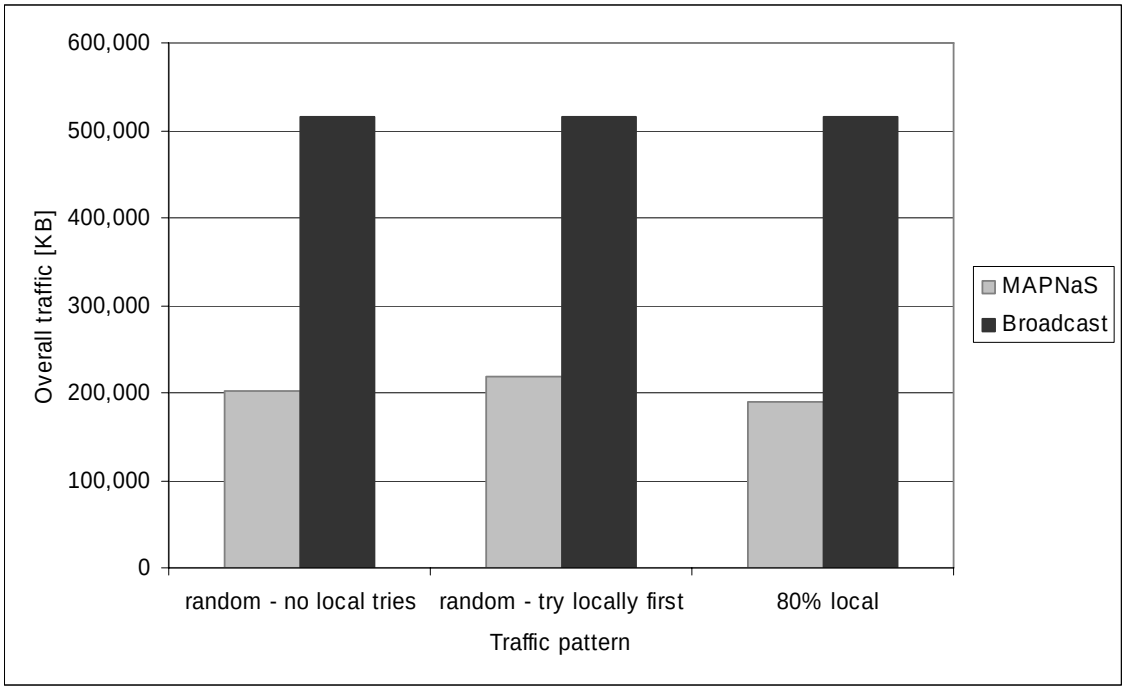

Figure 6.9 Overall traffic vs. traffic pattern.

unfavorable case, MAPNaS still significantly outperforms the broadcast-based approach by a factor of almost 3 (91% vs. 32%).

Figure 6.9 depicts the generated traffic. For the random traffic pattern, the figure with initial local lookups is slightly higher than the figure without initial local lookups due to the longer lookup paths (unsuccessful initial local lookup + global lookup). In a traffic pattern where nodes request local resources 80% of the time, however, the shorter lookup paths (local lookups will often be successful, thus there will be no need to start a global lookup) help lower the overall traffic by 12% compared to the random scenario without initial first lookups.

6.7 Related Work

As already described, numerous approaches have been proposed for service discovery in MANETs. Please refer to Section 3.4 for an overview of the various concepts employed in this area.

In the Internet, several systems have been proposed to build a completely distributed DNS using DHTs [2, 6, 8, 43, 61]. Similar in spirit to MAPNaS, the general idea, here, is always to store the entries under their hash keys in a DHT and to retrieve them using the key-based routing provided by the DHT. However, due to the usage of conventional DHTs, these systems are not well-suited for MANETs. It was the purpose of MAPNaS to demonstrate how to implement such DHT-based service discovery for MANETs.

6.8 Summary

MAPNaS provides efficient DHT-based service discovery for MANETs by using MADPastry as its DHT building block. Thus, instead of having a number of dedicated directory servers, every MAPNaS node serves both as a resource directory for certain resources and as a host of some resources (its own).

Through simulations, it has been shown that MAPNaS on top of MADPastry achieves significantly better lookup success rates than a naïve broadcast-based reference application while also producing significantly lower amounts of network traffic. For a comprehensive assessment of MAPNaS's performance, of course, further reference applications would clearly be required. This, however, is considered beyond the scope of this thesis. Instead, the implementation of MAPNaS serves two distinct purposes. First of all, it is a proof-of-concept for the practicality of MADPastry to serve as a general-purpose building block for distributed network applications in MANETs. Secondly, MAPNaS demonstrates how to build an efficient DHT-based name service in MANETs so that existing Internet-based distributed name service approaches could easily be adapted for the deployment in MANETs.