Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks

Himabindu Pucha Saumitra M. Das Y. Charlie Hu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907
{hpucha, smdas, ychu}@purdue.edu

Abstract

Distributed Hash Tables (DHTs) have proven to be a novel and efficient platform for building a variety of scalable and robust distributed applications like content sharing and location in the Internet. Similar to those in the Internet, distributed applications and network services in mobile ad hoc networks (MANETs) can potentially benefit from the deployment of a DHT. However, bandwidth limitations, node mobility, and multi access interference pose unique challenges to deploying such DHTs in MANETs.

In this paper, we first study how to efficiently implement DHTs in MANETs. We explore two disparate design options: the simple approach of directly overlaying a DHT on top of a MANET multi-hop routing protocol, and Ekta which integrates a DHT with a multi-hop routing protocol at the network layer. Second, we examine the efficiency of DHT substrates in supporting applications in MANETs by examining the performance of a resource discovery application built on top of Ekta with one that directly uses physical layer broadcast. Such a study answers the fundamental question of whether a DHT substrate can be more efficient in supporting applications than a physical layer broadcast-based protocol, since in MANETs, DHT protocols effectively rely on physical layer broadcast to discover and maintain routes.

1. Introduction

A mobile ad hoc network (MANET) consists of a collection of wireless mobile nodes dynamically forming a temporary network without the use of any existing network infrastructure or centralized administration. In such a network, nodes operate as both hosts and routers, forwarding packets for other mobile nodes that may not be within direct transmission range of each other. Such decentralized networks can enable flexible, infrastructure-less and robust data and service access to support ubiquitous computing environments. Due to the infrastructure-less environment, applications and network services in mobile ad hoc networks need to be designed to operate in a decentralized manner.

Recently, Distributed Hash Tables (DHTs) such as CAN, Chord, Pastry and Tapestry [19, 23, 22, 27] have been proposed as a novel platform for building a variety of scalable and robust distributed applications for the Internet, such as distributed storage systems [5, 21], application level multicast [3], and content-based full-text search [24]. A DHT substrate shields many difficult issues including fault-tolerance, locating objects, scalability, availability, load balancing, and incremental deployment from the distributed application designers. We argue that the DHT abstraction, if deployed in MANETs, could similarly provide an efficient way of constructing distributed applications and services. For example, applications such as file sharing and resource discovery can benefit from the distributed insert/lookup convergence provided by DHTs. However, DHTs have been designed for the Internet, and bandwidth limitations, node mobility, and multi access interference pose unique challenges to deploying DHTs in MANETs.

In this paper, we first study *how to efficiently support a DHT abstraction in highly dynamic mobile ad hoc networks*. We explore two design options: a layered approach which directly overlays a DHT on top of an existing multi-hop routing protocol for MANETs, and an integrated approach which integrates a DHT with a multi-hop routing protocol at the network layer and maximally exploits the interactions between the two protocols. In particular, the layered approach is implemented by laying Pastry [22] on top of DSR [12], and the integrated approach, implemented in Ekta, integrates Pastry and DSR at the network layer.

Our experimental results show that the integrated approach to supporting the DHT abstraction in MANETs used by Ekta is superior to the layered approach in terms of the number of data packets successfully delivered and the average delay in delivering the packets while incurring comparable routing overhead. This suggests that the efficient way of implementing a DHT in MANETs is by integrating the functionalities of the DHT into the routing layer as opposed to having two independent layers with minimal interactions.

Since Ekta fundamentally relies on flooding of packets to discover and maintain routes in supporting the DHT abstraction, a fundamental question is *whether applications built on top of Ekta will be more efficient than those directly built on top of physical layer broadcast*. To answer this question, in the second part of this paper, we perform a case study using resource discovery as a concrete application in MANETs. Using ns-2 simulations, we perform a detailed comparison between the performance of resource discovery built on top of Ekta with an approach that directly uses physical layer broadcast.

Our experimental results show that for the resource discovery application, the DHT-based approach consistently outperforms the broadcast-based approach for a wide range of application parameters. Specifically, the broadcast-based implementation incurs comparable routing overhead to that using Ekta for high inter-arrival time between resource queries. As the inter-arrival time decreases, the Ekta-based implementation incurs up to an order of magnitude lower overhead compared to the broadcast-based one. Furthermore, these results hold true for a wide range of mobilities. These results suggest that efficient peer-to-peer substrates such as Ekta provide a viable and efficient approach to building distributed applications in mobile ad hoc networks.

In the final part of the paper, we discuss implementation details of Ekta in the Linux operating system.

2. Ekta: An Efficient DHT Substrate for Mobile Ad Hoc Networks

In this section, we explore two opposite options in the design space of implementing a DHT in MANETs. In the first design, a DHT is directly layered on top of a multi-hop routing protocol, with minimum modifications to the routing protocol. This approach is thus similar to implementing a DHT in the Internet; it leverages the existing routing infrastructure for MANETs to the full extent. This approach, while consistent with the layered principle of the ISO model of networking, makes it difficult to exploit many optimization opportunities from the interactions between the DHT protocol and the underlying multi-hop routing protocol. For example, when the routing protocol is Dynamic Source Routing (DSR) [12] which uses caching to reduce routing overhead, it is difficult for the routing structures of the DHT and the route cache of DSR to coordinate with each other to optimally discover and maintain source routes.

Ekta adopts the opposite approach, that is, to fully integrate the functions performed by the DHT protocol operating in a logical namespace and by the MANET routing protocol operating in a physical namespace. The key idea of the integration is to bring the structured p2p routing protocol of the DHT to the network layer of MANETs via a one-to-one mapping between the IP addresses of the mobile nodes and their nodeIds in the namespace. With this integration, the routing structures of a DHT and of a multi-hop routing protocol, e.g., the route cache of DSR, are integrated into one structure which can maximally exploit the interactions between the two protocols to optimize the routing performance.

We note that although an overlay can be constructed in an ad hoc network, similarly to in the Internet, the purpose of this paper is not to motivate such an architecture. Since an ad hoc network is typically formed of nodes that collaborate with each other to enable communication among all the nodes, we believe it is rarely necessary to construct an overlay that consists of a subset of the nodes. In other words, unlike in the Internet where DHTs run on overlays, we propose to support DHT by all nodes in an ad hoc network.

2.1. Background

Since our study uses a concrete DHT, Pastry, and a representative routing protocol for MANETs, DSR, we first give a brief overview of DSR and Pastry in the following.

2.1.1. DSR DSR [12] is a representative multi-hop routing protocol for ad hoc networks. It is based on the concept of source routing in contrast to hop-by-hop routing. It includes two mechanisms, *route discovery* and *route maintenance*.

Route discovery is the process by which a source node discovers a route to a destination for which it does not already have a route in its cache. The process broadcasts a ROUTE REQUEST packet that is flooded across the network in a controlled manner. In addition to the address of the original initiator of the request and the target of the request, each ROUTE REQUEST packet contains a route record, which records the sequence of hops taken by the ROUTE REQUEST packet as it propagates through the network. ROUTE REQUEST packets use sequence numbers to prevent duplication. The request is answered by a ROUTE REPLY packet either from the destination node or an intermediate node that has a cached route to the destination. To reduce the cost of the route discovery, each node maintains a cache of source routes that have been learned or overheard, which it uses aggressively to limit the frequency and propagation of ROUTE REQUESTS. The route maintenance procedure monitors the operation of the route and informs the sender of any routing errors. If a route breaks due to a link failure, the detecting host sends a ROUTE ERROR packet to the source which upon receiving it, removes all routes in its cache that use the hop in error.

Both route discovery and maintenance benefit from optimizations such as overhearing routes and route errors made possible by the broadcast nature of the medium access environment.

We consider DSR with two cache designs: DSR-Path [2] which uses a *path cache* to store whole source routes, and DSR-Link [9] which uses a *link cache* to store individual links of routes to build a topological graph of the network. The graph potentially increases the effectiveness of the cache since it enables DSR to construct routes (using the graph) that were neither overheard nor discovered.

2.1.2. Pastry Pastry [22] is one of the several structured p2p routing protocols that implement the DHT abstraction. In a Pastry network, each node has a unique, uniform, randomly assigned nodeId in a circular 128-bit identifier space. Given a message and an associated 128-bit key, Pastry reliably routes the message to the live node whose nodeId is numerically closest to the key.

In a Pastry network consisting of N nodes, a message can be routed to any node in less than $\log_{2^b} N$ steps on average (b is a configuration parameter), and each node stores only $O(logN)$ entries, where each entry maps a nodeId to the associated node's IP address. Specifically, a Pastry node's routing table is organized into $\lceil \log_{2^b} N \rceil$ rows with $(2^b - 1)$ entries each. Each of the $(2^b - 1)$ entries at row n of the routing table refers to a node whose nodeId shares the first n digits with the present node's nodeId, but whose $(n+1)$th digit has one of the $(2^b - 1)$ possible values other than the $(n + 1)$th digit in the present node's nodeId. Pastry stores multiple candidates per routing table entry to increase availability. In addition to a routing table, each node maintains a leaf set, consisting of $L/2$ nodes with numerically closest larger nodeIds, and $L/2$ nodes with numerically closest smaller nodeIds, relative to the present node's nodeId. L is another configuration parameter. In each routing step, the current node forwards a message to a node whose nodeId shares with the message key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current nodeId. If no such node is found in the routing table, the message is forwarded to a node whose nodeId shares a prefix with the key as long as the current node, but is numerically closer to the key than the current nodeId. Such a node must exist in the leaf set unless the nodeId of the current node or its immediate neighbor is numerically closest to the key.

2.2. Design Options

In this section, we describe two opposite design approaches to implementing DHTs in MANETs, using a proximity-aware DHT Pastry and an on-demand MANET routing protocol DSR as concrete examples. The first approach directly overlays Pastry on top of DSR, whereas the second approach integrates the two protocols at the network layer.

2.2.1. Layered Approach In the layered approach, Pastry is directly layered on top of MANETs in the same way it is layered on top of the Internet. Pastry maintains its leaf set and routing table entries without source routes and DSR maintains source routes passively as per the demand of Pastry routing state.

However, a straightforward layering is not pragmatic, and three modifications are made to accommodate the shared medium access nature of MANETs: (1) Pastry's node joining process is modified to use expanding ring search for locating a bootstrap node to join the network; (2) The original Pastry uses an expensive "ping" mechanism with a delay metric to measure and maintain the proximity of nodes in its routing tables. We modified Pastry to use a hop count metric for proximity since in MANETs, delay is affected by many factors and has a high variability; (3) To reduce the cost of this proximity probing, we modified DSR to export an API that allows Pastry to inquire about the proximity values for nodes it is interested in. DSR can then use its cache to reply to "pings" from Pastry if there is a cached path to the node being pinged. In the absence of such a cached path, a ROUTE REQUEST is initiated by DSR.

2.2.2. Ekta: An Integrated Approach Ekta implements the DHT abstraction by integrating Pastry and DSR at the network layer which exploits optimizations made possible from close interactions between the two protocols. Previously, we proposed the design of DPSR in a position paper [8] as a network layer unicast routing protocol. Ekta and DPSR share the essence of tightly integrating Pastry and DSR at the network layer. However, unlike DPSR, Ekta does not use the DHT for unicast routing.

Node Addressing Ekta assigns unique 128-bit nodeIds to nodes in a MANET by hashing the IP addresses of the nodes using a collision-resistant hashing function such as SHA-1 [6].

Node State The structures of the routing table and the leaf set stored in each Ekta node are similar to those in Pastry. The difference lies in the content of each leaf set and routing table entry. Each entry in Ekta's leaf set and routing table stores a source route to reach the designated nodeId. As in Pastry, any routing table entry is chosen such that it is physically closer than the other choices for that routing table entry. This proximity awareness is continuously maintained by making use of the vast amount of indirectly received routes (from overhearing or forwarded messages) or by using prefix-based route discovery as described below in routing.

The size of the leafset (L) in Ekta is chosen to be 16. The parameter b of Pastry is a trade-off between the size of the populated portion of the routing table (approximately $\lceil \log_{2^b} N \rceil$*$(2^b - 1)$ entries) and the maximum number of hops required to route between any pair of nodes (i.e., $\lceil \log_{2^b} N \rceil$). The value of b is chosen as 4.

For efficiency, each routing table entry stores a vector of source routes to one or more nodes that match the prefix of that entry. Similarly, each leafset entry stores multiple routes to the designated node. The replacement algorithm used in each leaf set or routing table entry is Least Recently Discovered (LRD), disregarding whether a route is discovered directly or indirectly. When looking up a route from a leaf set or routing table entry, the freshest among the shortest routes in that 1-D entry is returned.

Routing In Ekta, a message with a 128-bit key is routed using Pastry's prefix-based routing procedure and delivered to the destination node whose nodeId is numerically closest to the message key. When a route lookup for the next logical hop returns a next-hop node from the leafset for which a source route does not exist, Ekta initiates a route discovery to discover a new source route. On the other hand, if the node selected as the next hop is from the routing table and does not have a route, a *prefix-based route discovery* is performed to discover routes to any nodes whose nodeIds match the prefix for that routing table entry. Note that each hop in the Ekta network is a multi-hop source route, whereas each hop in a corresponding Pastry network is a multi-hop Internet route.

Node Arrival/Departure Ekta uses a modified form of the Pastry join protocol to handle node arrivals. Each JOIN message is routed to the node whose nodeId is closest to the joining node. In contrast to Pastry, only this node responds with a JOIN COMPLETE message containing its leaf set. To maintain leaf set consistency, this node also notifies the members of its leaf set about the arrival of the new node using a broadcast flooding which records the path traversed as the packet is propagated. The leaf set members then send back acknowledgments using the recorded path (source route). Similarly, a node floods a LEAVE message with the leaf set members acknowledging the event. All the nodes receiving the LEAVE message remove this node from their routing tables.

Note that Ekta join and leave procedures are lightweight and incur message overhead only to maintain leaf set consistency which is required for DHT convergence. Unlike Pastry, no routing entry exchanges or proximity probing is carried out. Routing table entries are discovered on demand using low overhead prefix-based route discoveries and overhearing.

Optimizations Ekta inherits all of the optimizations on route discovery and route maintenance used by the DSR protocol. In addition, Ekta updates its routing table and leaf set using routes snooped while forwarding and overhearing packets, thus constantly discovering fresh and low proximity routes for the leaf set and the routing table entries. In addition to the "prefix-based view" of the routing table and the "neighbor-node view" of the leaf set, the Ekta routing structures can be viewed as two caches of source routes. These can be used to support unicast routing by Ekta whenever required by the application. For example, an *Insert* operation in a DHT-based application may travel over multiple hops in the nodeId space while an acknowledgment to the *Insert* could be efficiently unicast back to the originator.

2.3. Evaluation Methodology

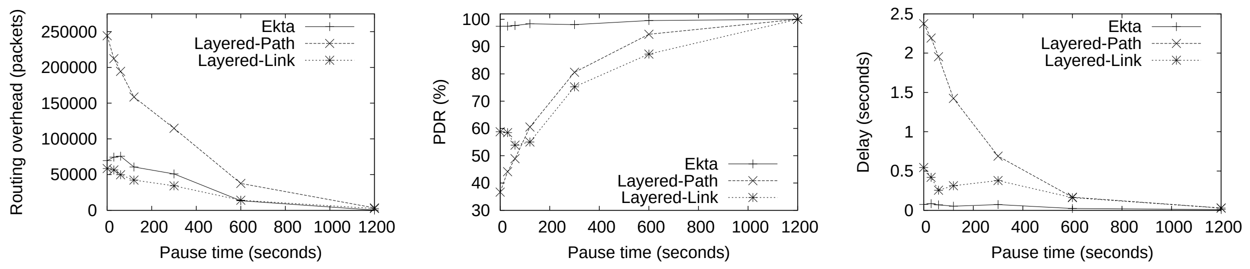
We use ns-2 [1] to evaluate the performance of both approaches to implementing DHTs. For the layered approach, we implemented *Pastry* as an application in ns-2 and ran it over DSR with a path cache and DSR with a link cache as studied in [9]: *DSR-PathGen64* and *DSR-LinkMaxLife*. The two versions are referred to as *Layered-Path* and *Layered-Link* in the rest of the paper. Like DSR, Ekta was implemented as a routing agent in ns-2.

The mobility scenarios are generated using a modified "random waypoint" model [26]. In our model, 50 nodes move at a speed uniformly distributed between 1-19 m/s in an area of 1500m x 300m. A wireless radio with 2 Mbps bit rate and 250m transmission range is used. The simulation duration chosen is 900s.

The communication pattern consists of 40 traffic sources, each initiating packets at the rate of 3 packets/second. Each packet has a 48-byte message body, prepended with a 128-bit key generated from hashing the message body. Thus, the effective packet payload is 64 bytes. This communication pattern models the traffic in a DHT-based storage system such as PAST [21].

To evaluate the steady state behavior of the DHTs, we use a cutoff time of 300s in a 1200s simulation, resulting in a duration of 900s over which the performance statistics are collected. This cutoff is chosen such that all connections have been initiated, all nodes have joined the network, and the instantaneous average speed of nodes in the network has stabilized to a steady state value [26].

The following metrics are evaluated for the routing protocols: (1) Routing overhead – The number of control packets transmitted, with each hop-wise transmission of a control packet counted as one transmission; (2) Packet delivery ratio (PDR) – The ratio of the data packets delivered to the *correct* destinations, i.e., nodes whose nodeIds are closest to the keys of the data packets, over the data packets generated by the traffic sources; (3) Average delay – The average end-to-end delay in routing a data packet.



(a) Routing overhead (b) PDR (c) Delay

Figure 1. Routing overhead, PDR and delay for varying mobility.

2.4. Performance Results

Figure 1(a) compares the routing overhead of Ekta and the two layered versions as the network mobility is varied. The routing overhead of Ekta is much lower than *Layered-Path* for all mobilities. The higher overhead in *Layered-Path* can be attributed to the following reasons:

(i) *Layered-Path* employs periodic routing table maintenance (every 250 seconds). These additional maintenance (non-data) packets and the route discoveries caused by them increase its overhead. In addition, probing the proximities of the exchanged nodes also adds to the routing overhead. In contrast, Ekta uses overhearing of routes from physically nearby nodes to maintain proximity of its routing table entries.

(ii) *Layered-Path* selects a node as the next overlay hop irrespective of whether there are routes to that node in the DSR route cache. This can cause unnecessary route discoveries. In Ekta, a node that has a valid source route is given preference and only when no such node exists, a prefix-based route discovery is issued.

(iii) *Layered-Path* selects a node as the next hop regardless of the relative freshness of its DSR source route compared to other candidates, since Pastry can not tell. This can lead to an increase in ROUTE ERRORS. We observed that for a pause time of 0s, Ekta incurred 9725 ROUTE ERRORS while *Layered-Path* had more than 18,000 ROUTE ERRORS. Ekta uses the Least Recently Discovered replacement to maintain source routes for each 1-D routing table entry and leaf set entry. Thus next-hop nodes are selected such that the routes to them are both fresh and short.

(iv) Although *Layered-Path* discovers candidates with better proximities for routing table entries via routing table maintenance, this approach does not refresh the proximities of the existing candidates. While this approach is suitable for the Internet where the proximity of nodes changes slowly, in a highly dynamic ad hoc network, it can lead to stale proximity information and consequent selection of distant nodes as next hops. This is exacerbated by the fact that source routes to these distant nodes may need to be discovered as well. In contrast, the proximities of candidates in Ekta are continually refreshed from forwarding and overhearing of packets.

A consequence of this high routing overhead is that the PDR of *Layered-Path* drops with increased mobility as shown in Figure 1(b). This is because the network capacity available for data packets is reduced. Additionally, packets traverse longer routes due to stale proximity selection, which increases the probability of route errors due to both the increased number of physical hops and the larger message header from longer source routes. In contrast, the PDR for Ekta remains largely constant with increased mobility.

A further consequence of the high routing overhead in the layered approach is that the routing delay of *Layered-Path* increases with increased mobility as shown in Figure 1(c). In contrast, the routing delay of Ekta remains almost constant.

Layered-Link which builds a topological graph of the network can construct routes it has neither discovered nor overheard, allowing it to have a lower rate of ROUTE REQUESTS and consequently lower routing overhead. However, these routes are constructed to potentially distant nodes and are frequently stale. It was observed in the simulations that although the number of ROUTE REQUESTS and consequently ROUTE REPLYS in *Layered-Link* are much lower than in *Layered-Path* and slightly lower than in Ekta, the number of ROUTE ERRORS in *Layered-link* are comparable to in *Layered-Path* and much higher than in Ekta. As a result, as the mobility increases, *Layered-Link* incurs a lower overhead than Ekta, but its PDR is much lower than that of Ekta.

In summary, Ekta is superior to both *Layered-Link* and *Layered-Path* in the number of data packets successfully delivered. Although *Layered-Link* maintains a lower routing overhead, the inability to exploit the interactions between the two protocols contributes to its low packet delivery ratio. This suggests that the correct way of implementing a DHT in MANETs is by integrating the functionalities of the DHT into the routing layer as opposed to having two independent layers with minimal interactions.

3. Application Case Study: Resource Discovery in MANETs

An efficient DHT substrate in MANETs such as Ekta can greatly ease the construction of distributed applications and services in MANETs by shielding many common and difficult issues such as fault-tolerance, object location, load balancing, and incremental deployment from the developer. In this sense, providing a DHT substrate in MANETs is even more significant than in the Internet, since these issues are especially challenging in a wireless, mobile environment. However, due to mobility in MANETs, any DHT substrate based on on-demand routing may trigger repeated flooding-based route discoveries to discover and maintain routes. In other words, a DHT-based application in MANETs may also experience many floodings of control packets. Thus, a fundamental question is whether applications built on top of the DHT such as Ekta will be as efficient as or more efficient than those directly built on top of physical layer broadcast. To answer this question, we perform a case study using resource discovery in ad hoc networks as a concrete application.

In the following, we first formally define the resource discovery problem in mobile ad hoc networks. We then present the design, analysis, and evaluation of two alternative approaches to implementing resource discovery: one directly built on top of physical layer broadcast, and one built on top of Ekta.

3.1. Resource Discovery in MANETs

Pervasive wireless ad hoc networks are comprised of a variety of heterogeneous devices with varying energy resources, capabilities, and services to offer. It is natural for such systems to rely on peer cooperation to efficiently use each other's resources. Examples of resource discovery include discovering nodes with GPS devices so other nodes can approximate their location, collecting sensed information from mobile sensors, contacting location and directory servers, and locating people with specific capabilities in disaster relief and battlefield networks. For instance, a resource lookup in a platoon of soldiers or in a team coordinating disaster relief can be of the form *"Find the closest medic"*. These examples show that efficient discovery of resources that meet certain requirements in an ad hoc network is of great importance to building a variety of distributed applications in ad hoc networks.

We assume a set R of unique resources are present in a network with N nodes. Each resource $R_i \in R$ is replicated with some probability q_i on each node in the network. We assume that each resource R_i in the ad hoc network has a well known name (e.g., GPS) or identifier. Resource ownership information once discovered is not cached since in a MANET, resources could be dynamic (nodes join and leave), and their availability is time-varying, i.e., depending on their current usage.

Discovering a resource includes three steps: retrieving a list of a set of nodes $N_i \subseteq N$ that own the requested resource R_i, selecting a physically close owner of R_i from this list, and finding a route to that owner. This operation should be completed with low overhead, high success rate, and low delay.

3.2. Design Options

For our case study, we implemented the resource discovery application in ns-2 based on two decentralized approaches, unstructured (Gnutella-like) and structured (DHT-based). Both versions are modeled after the Service Location Protocol framework [7]. The detailed wireless simulation capabilities of ns-2 allow us to examine the tradeoffs between the two versions when running on a MANET.

DSR-RD: Resource Discovery using DSR *Route Requests* The first version of the resource discovery application is integrated with the DSR routing protocol. It essentially uses physical layer broadcast augmented with source routing to perform resource discovery as follows. We modified the route discovery process of DSR to support discovering resources using resource specific identifiers. Each node transmits RESOURCE REQUESTS (similar to ROUTE REQUESTS of DSR) and each node that does not own the resource requested, rebroadcasts the RESOURCE REQUESTS after encoding its IP address into the source route. If a node in the network owns that resource, it responds with a service reply (RESOURCE REPLY) that is unicast back to the requester similar to the ROUTE REPLY of DSR. RESOURCE REQUESTS contain sequence numbers to ensure that the overhead incurred is at most N (network size) packets. The remaining overhead is the number of RESOURCE REPLY transmissions which is determined by the degree of replication of the resource being requested.

Ekta-RD: Resource discovery using Ekta The second version builds the resource discovery application on top of the Ekta DHT substrate. In the following this version is referred to as *Ekta-RD*. Ekta provides three DHT APIs, *route(Message, Key)*, *route(Message, IP Address)* and *broadcast(Message, Broadcast Address)*, as well as an additional API *Proximity(IP Address)* which are used by the application in various stages of its operations. The *Proximity(IP Address)* interface returns the the hop distance of the node specified by looking up the locally cached routes. If no such route is cached, Ekta returns null.

Ekta-RD simply relies on Ekta to route a RESOURCE REQUEST to the correct directory agent and receive a reply. When a resource is required, *Ekta-RD* hashes the resource identifier into a *key* and invokes *route(Message, key)* of Ekta to send this request to a node $N_i \in N$ whose *nodeID* (hash of network address) is numerically closest to this *key*. This node N_i is the directory agent for the mapped resource and contains previously inserted information for this resource. It replies to the requester with a list of nodes that own the resource in the network. The requester then finds the closest node out of this list using Ekta 's *Proximity(IP Address)* interface and contacts the chosen node to use the resource. If Ekta cannot determine the proximity of any node in the list returned in the RESOURCE REPLY, the application randomly selects a node out of the list and contacts that node to use the resource. This will trigger a ROUTE REQUEST for that random node by Ekta.

3.3. Analysis

In this section, we analyze the overhead incurred by *DSR-RD* and *Ekta-RD*. We define the following parameters. P is the average number of hops between any two nodes. λ is the average number of resource requests per node. Note that $\lambda = \frac{Simulation\ time}{Average\ interarrival\ time}$. N is the number of nodes in the network. The average degree of replication of any resource is q. In *DSR-RD*, a single resource request triggers N packet transmissions. A total of $q \cdot N$ resource replies are received per request, each causing an overhead of P transmissions. Thus the total overhead incurred by each node is given by $\lambda \cdot N + \lambda \cdot q \cdot N \cdot P$ and the corresponding total overhead is

$$X_{DSR-RD} = \lambda \cdot N^2 + \lambda \cdot q \cdot N^2 \cdot P \quad (1)$$

In *Ekta-RD*, each resource request is routed in $\log_{2^b} N$ overlay hops and a reply is unicast back in P transmissions. The overhead per node assuming all routes used were cached and valid is $\lambda \cdot \log_{2^b} N \cdot P + \lambda \cdot P$. However, in MANETs routes break, and new routes need to be discovered using flooding. In the worst case, when all routes in a transmission sequence are invalid, $\log_{2^b} N + 1$ floodings will be initiated. If P_v is the average probability that a route is cached and valid and P_b is the average probability that a route is not cached or cached but stale, then the total overhead in Ekta is given by

$$\begin{aligned} X_{Ekta-RD} &= N \cdot P_v(\lambda \cdot \log_{2^b} N \cdot P + \lambda \cdot P) + \\ &\quad N \cdot P_b(\lambda \cdot \log_{2^b} N \cdot P + \lambda \cdot P + \\ &\quad \lambda \cdot (\log_{2^b} N + 1) \cdot N) \quad (2) \end{aligned}$$

From the above equations, we can infer that overhead in *DSR-RD* is independent of the mobility whereas *Ekta-RD* will have increasing overhead as mobility is increased. With increasing N, the overhead grows as $O(N^2)$ for *DSR-RD* and as $O(N \cdot \log_{2^b} N)$ for *Ekta-RD* (since $P_b < P_v$ for most practical scenarios). Another observation is that as the inter-arrival time increases, λ decreases and thus the overhead for both approaches decreases.

3.4. Evaluation Methodology

The simulation parameters used in this evaluation are similar to the parameters outlined in Section 2.3. As before, mobility scenarios are generated using the modified "random waypoint" model [26]. The nodes move at a speed uniformly distributed between 1-9 m/s. The number of unique resources in the network, i.e., the size of the set R, is chosen to be equal to the number of nodes in the network. Each resource R_i is owned by on average 10% of the nodes in the network.

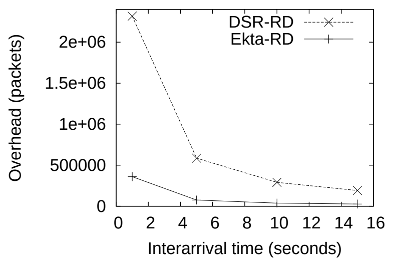
The communication pattern of the resource discovery application consists of resource requests that are modeled using a Poisson arrival process. The average inter-arrival time (ΔT) of these requests is varied from 1 to 15 seconds. Each resource request aims at locating a resource R_i randomly selected from the set R. Hashing the identifier of resource R_i generates a unique 128-bit key. This key is used by Ekta to route the resource request message. We use areas of sizes 1500m x 300m, 2000m x 500m, 3000m x 600m for networks of size 50, 100, and 150, respectively.

The following metrics are measured to compare the two implementations: (1) Overhead – The total number of packets including the control packets transmitted for the resource discovery operation, with each hop-wise transmission of a packet counted as one transmission; (2) Success ratio – The ratio of the number of resource requests resolved over the number of resource requests generated by the application. We do not measure the resolution latency, since there is no easy way to compare the two implementations in a fair manner. The delay for the first reply to be received in *DSR-RD* is expected to be much shorter than receiving the single reply in *Ekta-RD* containing the whole list of owners. On the other hand, the average delay in receiving all the replies for a request in *DSR-RD* is expected to be comparable to that in *Ekta-RD*.

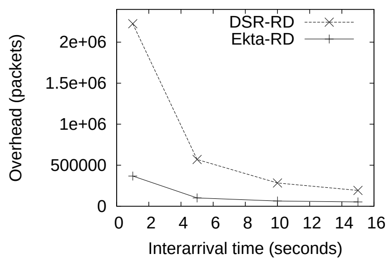
3.5. Performance Results

In the following, we compare the performance of the two implementations of the resource discovery application with varying request rate, mobility, and network size.

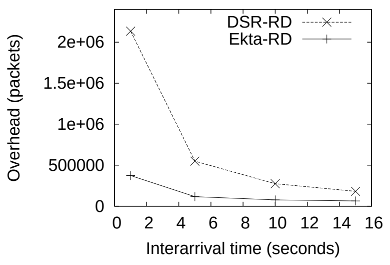
3.5.1. Effects of Request Rates and Mobility In this experiment, the average inter-arrival time of the requests generated by each node is varied from 1 second to 5, 10, and 15 seconds. As the inter-arrival time ΔT increases, the number of resource queries generated per node λ (and thereby



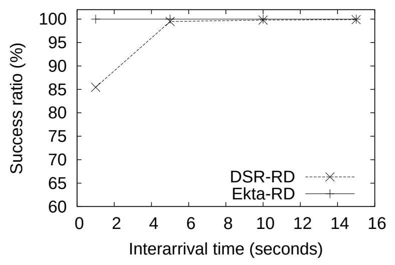
(a) Overhead (pause time 1200s)



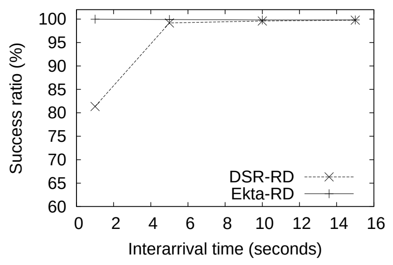
(b) Overhead (pause time 300s)



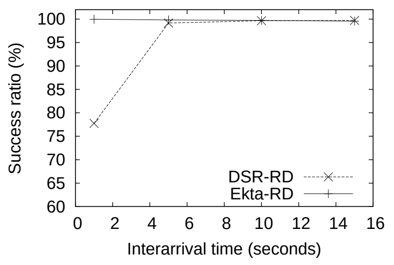
(c) Overhead (pause time 0s)



(d) Success ratio (pause time 1200s)



(e) Success ratio (pause time 300s)



(f) Success ratio (pause time 0s)

Figure 2. Overhead and success ratio comparison as the inter-arrival time increases.

in the overall system) decreases. Thus the larger the value of ΔT, the lesser the congestion in the network. The results of this experiment are shown in Figure 2. The following observations can be made from Figure 2.

First, the overhead for *DSR-RD* is inversely proportional to the inter-arrival time as seen in Figures 2(a), 2(b), and 2(c) for all the pause times. This is because the overhead of *DSR-RD* is proportional to λ and λ is inversely proportional to the inter-arrival time as seen in Section 3.3.

Second, Figures 2(a), 2(b), and 2(c) show that the overhead for *Ekta-RD* also decreases as the the inter-arrival time increases for all pause times. In addition to the overhead reduction due to decreasing λ, further reduction is caused by reduced congestion in the network resulting in lower probability of routes being mistakenly invalidated (which can increase P_b in Equation (2)).

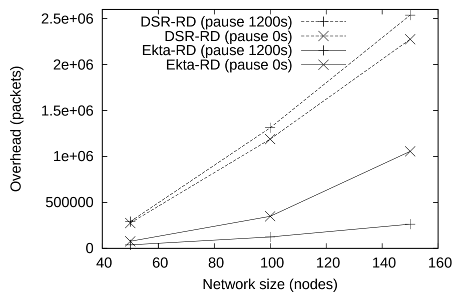
Third, the ratio between the overhead of *DSR-RD* and *Ekta-RD* remains largely constant as the inter-arrival time increases for each fixed mobility. However, as the mobility increases, the gap between the overhead of the two versions narrows. This is because the overhead of *DSR-RD* is largely independent of the mobility, while *Ekta-RD* is more likely to experience broken routes from increased mobility (which increases P_b).

Finally, the success ratios for both implementations approach 100% except for the case of inter-arrival time being 1 second as observed in Figures 2(d), 2(e), and 2(f). The drop in success ratio for *DSR-RD* can be explained as follows. When the inter-arrival time is 1 second, the routing overhead for *DSR-RD* is so large that the increased congestion and multi-access interference in the network cause packets to be dropped, reducing the success ratio.

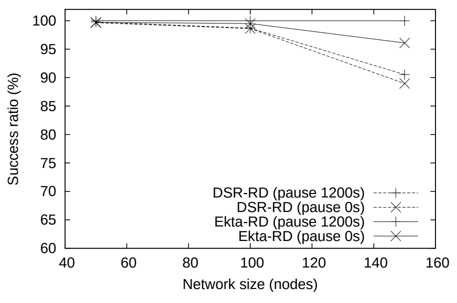
3.5.2. Effects of Network Size In this experiment we vary the network size among 50, 100, and 150 nodes while keeping the average inter-arrival rate constant at 5 seconds. We perform the experiments for a static network (pause time 1200s) and a highly mobile network (pause time 0s).

Figures 3(a) and 3(b) depict the overhead and success ratio as the network size is varied. First, for a static network, the overhead of *Ekta-RD* scales slowly with increasing network size ($O(N \cdot \log_{2^b} N)$) as compared to *DSR-RD* ($O(N^2)$) and as a result the relative performance gap increases with increased network size. At 150 nodes, the overhead of *DSR-RD* grows so large that its success ratio drops to 90% and thus 10% of resource requests do not succeed. In contrast, *Ekta-RD* maintains close to 100% success ratio throughout all network sizes.

Second, when the network is changed from static to highly mobile, Figures 3(a) and 3(b) show that the overhead and success ratio for *DSR-RD* remains largely unchanged for all network sizes. This is again because of the broadcast nature of the implementation. In contrast, *Ekta-RD* suffers significantly increased overhead and reduced success ratio when the network size increases. At 150 nodes, the success ratio of *Ekta-RD* drops by 4%. This occurs because the in-



(a) Overhead



(b) Success ratio

Figure 3. Overhead and success ratio comparison as the network size increases.

creased network size coupled with high mobility causes an increased number of broken routes and consequently lower success ratio. Despite the drop in success ratio, *Ekta-RD* continues to outperform *DSR-RD* as the network is scaled from 50 nodes to 150 nodes.

4. Implementation Details

In this section, we discuss the implementation details of Ekta on our ad hoc testbed, which comprises of 5 laptops and 15 PDAs running Linux with Orinoco wireless cards which support 802.11 ad hoc mode. Ekta is implemented as a user-space library (*libEkta*) that implements the *route(msg,key)* DHT API. It operates on a well known application specified port and implements a routing table, a leaf set, and functionalities to perform prefix/route requests and replies as described in Section 2.2.2, as well as request timers and outstanding packet buffers. Target applications can link to this library to use the DHT API.

Figure 4 depicts the steps taken in routing a message in Ekta. (1) On receipt of a message to route, the application hashes the message and calls the *route(msg,key)* API of *libEkta*. (2) Ekta then performs a lookup for the next logical hop of the packet based on the message key and uses this as the IP destination. The packet is then sent out so that transport layer and IP headers can be added. (3) After the IP header is added, the locally generated packet is captured by the LOCAL_OUT hook of the Netfilter framework. This is done with the help of the Source Routing kernel module (SRM) depicted in the Figure. All packets are queued up for user-space processing by Ekta. We note that many ad hoc protocol implementations also incur this overhead of crossing the user-kernel boundaries twice [16, 4]. (4) Ekta searches for a route to the IP destination (next logical hop) of each packet received from the SRM module. If such a route does not exist, the packet is buffered and a prefix/route discovery is sent. (5) If a route exists, Ekta inserts the Ekta header (shown in Figure 5) and copies the source route to the Ekta option header, both in between the IP and transport layer headers. It then sends the packet to the next physical hop from the source route using a raw socket.

The Ekta packet structure is similar to that of DSR [11] and consists of a fixed 4-octet Ekta header followed by a sequence of 0 or more Ekta options. The Ekta header has a next protocol field and a length field (total length of the option headers). First, the (next transport) protocol field from the IP header is copied into Ekta header's next protocol field. Then a code unique to Ekta is copied into the IP protocol field. The IP length is then updated, and the checksum recalculated. Each Ekta control packet has its own option code and related option header.

The message now travels to the next physical hop on the source route towards the next logical hop. Similar to in the Internet, we want the packet to be forwarded to the next logical hop without involving the DHT protocol. (6) The SRM kernel module in the next physical hop captures the packet using the Netfilter PRE_ROUTING hook. (7) It then extracts the next hop from the source route in the packet and forwards the packet on to the next physical hop. (8) Additionally, it extracts the header information and queues it up for post processing by Ekta to snoop on routes and update its routing table. This provides the separation of complex Ekta operations from packet forwarding along the physical hops and allows physical packet forwarding to be performed quickly.

The packet is forwarded until it reaches the next logical hop, detected when the SRM module finds that the source route destination is the current node. (9) The SRM module then strips and buffers the Ekta header for source route snooping (10), updates the IP protocol field, and allows the packet to undergo normal stack traversal. (11) The packet is demultiplexed based on the port number and protocol to Ekta. (12) If Ekta finds that the key of the message is closest to the hash of current node's IP address it sends the message to the application. Otherwise it restarts the procedure of sending the message to a logical hop (Step (2)).

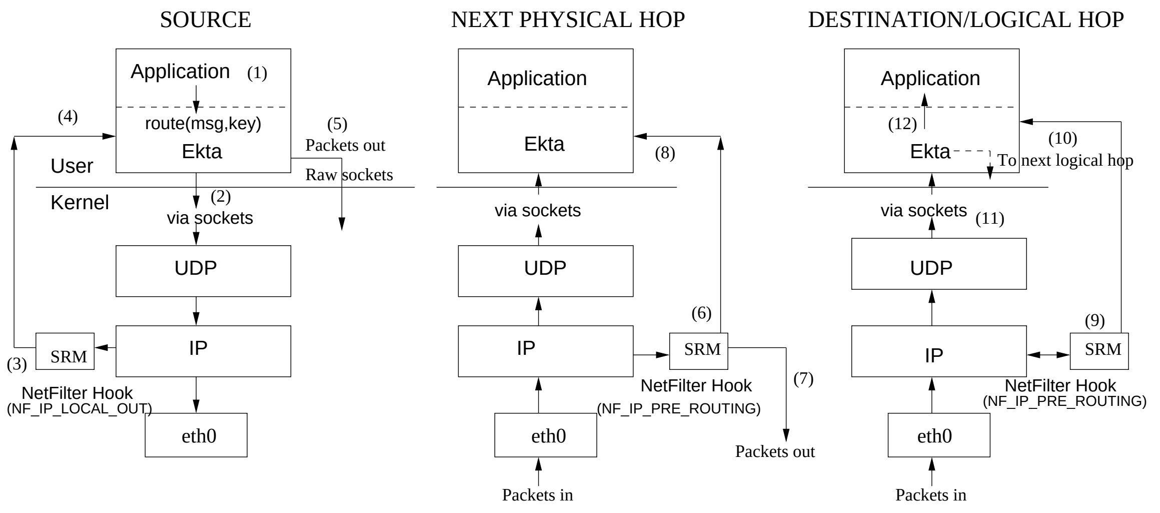


Figure 4. Ekta software architecture.

	Ekta Header			Ekta Options					Payload	
IP Header	Next Protocol	Reserved	Length	Option Type	Option Length	Option Data	....	Transport Header	Ekta Key	Message

Figure 5. Ekta packet structure.

Parts of our design are inspired by other work on ad hoc protocol implementation (e.g. [13, 16]). Our future work involves the following additions and improvements: (1) support to remove the double copying between user and kernel space; (2) support for multiple applications linked to *libEkta* running on each node and the associated demultiplexing; and (3) support for integrating unicast functionality into the Ekta module since the prefix based table can be used as a cache of routes. In the current design, an ad hoc unicast routing protocol runs independently from Ekta. This does not allow sharing of routing information among the protocols even though both could benefit from it.

5. Related Work

DHTs in MANETs Although there has been no previous work on supporting DHTs in MANETs without location information, several studies have proposed implementing DHTs with GPS support.

The geographic location system (GLS) in GRID [15] is a scalable location service that performs the mapping of a node identifier to its location. GLS can be combined with geographic forwarding to implement unicast. The implementation of GLS effectively provides a DHT interface; it routes a message with a nodeId Y to a node whose nodeId is closest to Y. However, GLS requires both GPS support as well as building a distributed location database. In [20], the authors proposed a geographic hash table that is inspired by DHTs but for data centric storage in sensornets. Like GLS, GHT requires GPS support, but unlike GLS, a GHT functions without a location database.

Resource (Service) Discovery in MANETs A large body of work exist on resource discovery for MANETs (e.g., [25, 14, 10]). Different from these work, the focus of Ekta-RD presented in this paper is to demonstrate DHTs as a general purpose substrate for building distributed applications in MANETs.

6. Conclusions and Future Work

Previous work in deploying p2p applications in MANETs have focused on modifying specific applications to take advantage of the rich connectivity of the broadcast medium. In contrast, our approach is to provide an efficient p2p substrate in MANETs over which diverse applications could be easily built and Internet-based p2p applications ported. In particular, we have studied the fundamental questions of how to efficiently implement DHTs in MANETs and whether applications built on top of such an DHT can be more efficient than those directly built on top of physical layer broadcast.

It remains interesting to see how to integrate prefix-based Pastry routing with other MANET routing protocols. The use of hop-by-hop routing as in AODV [18] would require that all nodes along a route to the next overlay hop maintain this route. This implies that these intermediate nodes need to have a prefix match with the destination nodeId. With source routing, however, the intermediate physical hops need not share any prefix with the destination nodeId.

It is also interesting to see how to efficiently integrate other DHTs such as CAN and Chord with MANET routing protocols. A Chord-based DHT would be similar to Ekta in that each node would store with its successor list and finger table a list of source routes to the corresponding nodeIds. However, since the routing table entries for Chord are required to refer to specific points in the Id space, proximity-aware selection of overlay hops would be less flexible than Pastry. A similar problem would exist in a CAN-based DHT.

Acknowledgments

We thank our shepherd Andrew Campbell and the anonymous reviewers for their helpful comments. This work was supported in part by NSF grant ANI-0338856.

References

[1] L. Breslau. et al., Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[2] J. Broch. et al., A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of ACM MobiCom*, October 1998.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *Proc. of ACM SOSP*, October 2003.

[4] I. D. Chakeres and E. M. Belding-Royer. AODV Routing Protocol Implementation Design. In *Proc. of WWAN*, March 2004.

[5] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP*, October 2001.

[6] FIPS 180-1. Secure Hash Standard. Technical Report Publication 180-1, Federal Information Processing Standard (FIPS), NIST, US Department of Commerce, Washington D.C., April 1995.

[7] E. Guttman. Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing*, 3(4):71–80, 1999.

[8] Y. C. Hu, S. M. Das, and H. Pucha. Exploiting the synergy between peer-to-peer and mobile ad hoc networks. In *Proc. of HotOS-IX*, May 2003.

[9] Y.-C. Hu and D. B. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *Proc. of ACM MobiCom*, August 2000.

[10] J.B.Tchakarov and N.H.Vaidya. Efficient content location in wireless ad hoc networks. In *Proc. of IEEE MDM*, January 2004.

[11] D. Johnson, D. Maltz, and Y. Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks, April 2003.

[12] D. B. Johnson and D. A. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Kluwer Academic, 1996.

[13] V. Kawadia, Y. Zhang, and B. Gupta. System services for ad-hoc routing : Architecture, implementation and experiences. In *Proc. of MobiSys*, May 2003.

[14] U. C. Kozat and L. Tassiulas. Network layer support for service discovery in mobile ad hoc networks. In *Proc. of IEEE INFOCOM*, April 2003.

[15] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *Proc. of ACM MobiCom*, August 2000.

[16] H. Lundgren and E. Nordstrm. Aodv simulation code (uppsala university), 2004. http://www.docs.uu.se/docs/research/projects/scanet/aodv/.

[17] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of ACM SIGCOMM*, August 1994.

[18] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of IEEE WMCSA*, February 1999.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *Proc. of ACM SIGCOMM*, August 2001.

[20] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash Table for Data-Centric Storage in SensorNets. In *Proc. of 1st ACM WSNA*, September 2002.

[21] A. Rowstron and P. Druschel. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. of ACM SOSP*, October 2001.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, November 2001.

[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. of ACM SIGCOMM*, August 2001.

[24] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proc. of ACM SIGCOMM*, August 2004.

[25] D. Tang, C.-Y. Chang, K. Tanaka, and M. Baker. Resource discovery in ad hoc networks. Technical Report CSL-TR-98-769, Stanford University, 1998.

[26] J. Yoon, M. Liu, and B. Noble. Random waypoint considered harmful. In *Proc. of IEEE INFOCOM*, April 2003.

[27] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-area Location and Routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.