Towards Efficient Range Queries in Mobile Ad Hoc Networks using DHTs

Thomas Zahn[1,2]
zahn@inf.fu-berlin.de

Georg Wittenburg[2]
wittenbu@inf.fu-berlin.de

Jochen Schiller[2]
schiller@inf.fu-berlin.de

INRIA[1]
Rocquencourt, France

Institute of Computer Science[2]
Freie Universität Berlin

ABSTRACT
Recently, Distributed Hash Tables (DHT) explicitly designed for the use in MANETs have been proposed. Thus, many DHT-based distributed network applications from the domain of the Internet can be expected to be efficiently ported to MANETs. While the exact key lookups provided by such DHTs might be sufficient for many applications, range queries are often a desirable feature in wireless ad hoc networks (e.g. in sensor networks). However, the implementation of range queries using DHTs is a non-trivial task.
In this paper we present a straight-forward implementation of Distributed Segment Trees as proposed in [4] on top of MADPastry [3] to provide DHT-based range queries for MANETs. The main goal of this work is to gain a first insight into the question whether DHT-based approaches for range queries are feasible in MANETs. First experimental results indicate that DHTs can indeed enable efficient range queries in MANETs.

Categories and Subject Descriptors
C.2.4 **[Distributed Systems]**

General Terms
Design, Algorithms, Experimentation

Keywords
DHTs, MANETs, range queries, Distributed Segment Trees

1. INTRODUCTION
With the continuous proliferation of ever more powerful mobile devices, larger (> 50 nodes) mobile ad hoc networks (MANETs), in which nodes communicate with each other wirelessly and without any pre-existing, central infrastructure, are becoming realistic. Having such larger MANETs, it would certainly be interesting to implement distributed network applications as they are known from the Internet such as name services, messaging systems or storage systems for MANETs as well.
In the domain of the Internet, a large number of those distributed network applications have been efficiently built on top of Distributed Hash Tables (DHTs, e.g. [2]). However, conventional DHTs have been designed primarily for the Internet and their dependency on reliable physical routing renders a direct deployment in MANETs unlikely to be successful (see e.g. [3]). For this reason, several systems have been most recently designed that provide DHT-like routing explicitly for MANETs (e.g. [1, 3]). With these DHT substrates, many of the aforementioned distributed network applications can be expected to be ported for the use in MANETs in a straight-forward way.
The general concept of a DHT is to enable efficient key-based routing. For this purpose, both participating nodes and the objects stored in the network are assigned hash keys from the same key space by hashing, for example, a node's MAC address and an object's name. A packet is then routed by the DHT based on a key (e.g. the key of an object that is to be inserted into the network) until it is eventually delivered to the node that is responsible for the packet's key. A node responsible for a given key is typically defined as the node whose own hash key is numerically closest to the given key among all live nodes in the network.
With their efficient key-based routing, DHTs represent a very suitable building block for numerous distributed applications. Nonetheless, DHTs exhibit a key weakness: they require *exact* key lookups. While this is sufficient for applications that, for example, store and retrieve resources under their names (or, more accurately, under the hash keys of their respective resource names), it is a non-trivial task to implement *range queries* – i.e. queries that return all values within a certain interval – with DHTs. In the case of a discrete query interval [a, b], the greedy approach using a DHT would be to generate the hash key for each value in [a, b] and issue a request for each such key (note that two succeeding values will hardly ever also have succeeding hash keys). Especially for larger query ranges, the traffic incurred by this greedy approach can easily outweigh that produced by a simple network-wide query broadcast. For non-discrete query intervals, the greedy approach would fail altogether.
Range queries, however, are often a desirable feature in distributed applications. Consider, e.g., a MANET in which the nodes use a DHT to advertise the number of resources they are sharing. Participating nodes might be interested in finding those nodes that share at least 100 but no more than 1,000 resources. In the context of wireless ad hoc networks, another conceivable example are wireless sensor networks (WSNs). In a large WSN, the sensor nodes could use a DHT to publish their current sensor readings and an application might, for instance, be interested in all current temperature readings between 0°C and 30°C.
To address this shortcoming, we present a DHT-based approach for range queries in MANETs that uses a Distributed Segment Tree as proposed in [4] on top of MADPastry [3], a locality-aware DHT substrate explicitly designed for MANETs. To the best of our knowledge, DHTs have not been previously used to enable

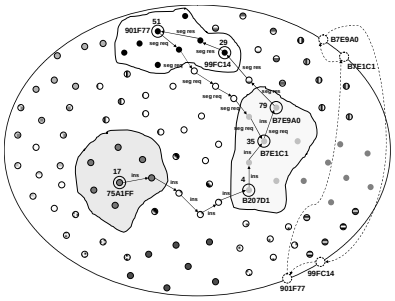

Figure 1. Segment insert, request and response.

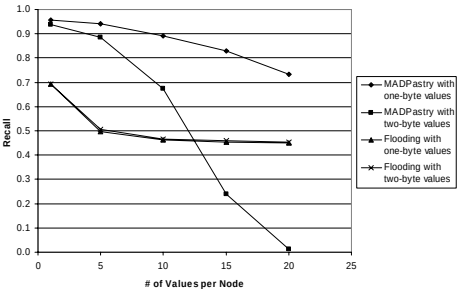

Figure 2. Values per node vs. recall.

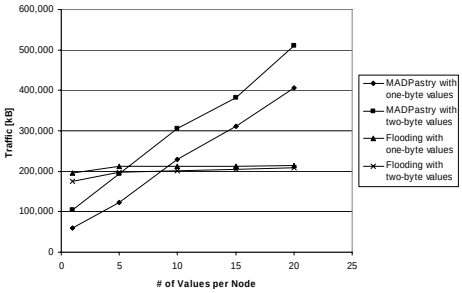

Figure 3. Values per node vs. traffic in kB.

range queries in mobile ad hoc networks. Specifically, our goal is to provide a first insight into the more general question whether DHT-based approaches for range queries are suitable for MANETs, or whether one would be better off using a simple broadcast approach.

2. DHT-BASED RANGE QUERIES

For our DHT-based range query approach, we use a straight-forward implementation of a DST on top of MADPastry. A DST is a balanced binary tree where each node represents a segment of the entire possible value range. The two child nodes of each inner DST node equally divide their parent's segment into two halves with the root node being responsible for the entire value space (please see [4]). Each DST node is now mapped to exactly one MADPastry node. For this purpose, the segment that a DST node holds is hashed into MADPastry's virtual ID space. The MADPastry node whose own virtual ID is numerically closest to the segment's hash key becomes responsible for that segment and stores all values (along with the providers' node addresses) that fall into the segment.

Inserts. When a node wants to insert a new value (or reinsert an existing one), it first determines all DST nodes whose segments cover the respective value. Note that, given the maximum possible value, each node can easily compute the DST locally as it is merely a virtual data construct. Having determined the matching DST nodes, the node will send an insert message containing the (new) value towards the respective hash keys using MADPastry which will eventually deliver the insert messages to the corresponding nodes currently responsible for the various segments. Clearly, the number of necessary insert messages per new value is equal to $\lceil \log_2(m+1) \rceil + 1$, where m is the maximum possible value (we assume 0 to be the minimum value).

Range Queries. When a node wants to find all values in a certain interval [a, b], it first uses the range splitting algorithm described in [4] to compute the minimum set of DST nodes whose combined segments exactly expand the range [a, b]. Again, this computation can easily be performed locally by each node. Following this computation, the node sends a segment request towards the respective DST nodes' hash keys using MADPastry. Upon reception of such a segment request, a node will reply with a segment response message that will contain all (value, node address) tuples that were stored under the requested segment.

Handovers. Unlike in the Internet-based original paper [4], due to node mobility, a MADPastry node might move out of its cluster and decide to join a new cluster by assigning itself a new virtual ID whose prefix matches that of the new cluster. Obviously, in such an event, the node would no longer be responsible for its stored DST nodes as its new virtual ID will most likely not be numerically closest to the respective DST nodes' hash keys. Therefore, whenever a node changes its overlay ID or after a certain period has expired, it checks for each of its stored DST nodes whether it knows another node whose virtual ID is numerically closer to the hash key of a stored DST node. If so, a handover packet containing all (value, node address) tuples of the respective DST node is sent towards the DST node's hash key and will eventually be delivered by MADPastry to the node now responsible for that hash key. Upon reception of an acknowledgement from that node, the node will remove the DST segment from its local storage.

Example. Figure 1 shows an example of a value insert. Suppose node 17, whose virtual ID is `75A1FF`, has a new value 7. Node 17 will first determine all necessary DST nodes that need to be contacted. One of those DST nodes might, e.g., be the node with segment [6, 7] whose hash key might be hash([6, 7]) = `B7E997`. Along with the other necessary insert messages, node 17 will then send an insert message towards that hash key (`B7E997`), which will eventually be delivered by MADPastry to node 79 whose virtual ID `B7E9A0` is numerically closest to the hash key. Analogously, Figure 1 also shows node 51 who is about to start a range query for [5, 7]. The range query might be split into the segments [5, 5] and [6, 7]. Node 51 will then send a segment request to both hash keys. For the segment [6, 7], e.g., node 51 will of course obtain the same hash key (`B7E997`) that node 17 obtained for its insert. Hence, node 51 will send a segment request towards that hash key that MADPastry will first deliver to node 35 who will forward the packet to node 79. Node 79 will reply with a segment response message that (at least) contains the tuple (7, node 17) and that will be routed back to node 51 by MADPastry using node 51's sender virtual ID (`901F77`).

3. EXPERIMENTAL RESULTS

In order to evaluate the architecture proposed in the previous section, we conducted a series of simulations using the ns-2 network simulator. We considered a network of 100 mobile nodes moving at 1.4 m/s according to a random waypoint model with a pause time of 0s (constant node movement) over an area of 1000m × 1000m. The transmission range of all nodes was fixed to 250 m and they used 802.11 for medium access.

Our main focus of interest was to establish the performance and associated cost of range queries in MANETs. To this end, we define performance as recall of queries, i.e. as the ratio of the number of values reported back to the querying node and the total number of matching values available in the entire network. We measure cost in terms of total network traffic incurred by these queries. Applications that use range queries directly depend on the number of values that can be stored in the network and the range

of possible values. We address this by varying the number of values provided by each node and by conducting simulations for both one- and two-byte values.

At the beginning of our simulations, each node generates a certain number of random values (between 0 and 255 for one-byte values, or between 0 and 65535 for two-byte values). We considered 1, 5, 10, 15, and 20 values per node. Each simulation run lasted one simulated hour. In the first 2 (simulated) minutes of the simulation, the MADPastry network is bootstrapped. To setup the DST, during the next first 5 (simulated) minutes of the simulation, nodes insert their values into the network (the inserts are uniformly distributed over this period). Recall that each value that a node provides needs to be inserted once at each level of the DST (see Section 0), i.e. for one-byte values, each value will incur 8 insert messages. As the depth of the DST doubles for two-byte values, we employed a 10-minute insertion period for those scenarios. Following that insert period, for the remainder of the simulated hour, each node generates a random range query (the interval is uniformly distributed between 0 and the respective maximum value) every 30 seconds.

As reference application for the evaluation, we also implemented range queries using a simple broadcast-based approach: Each range query is sent to all nodes in the network using AODV broadcast, and each node replies directly to the querying node by sending a list of matching values that are locally available. No reply is sent if no matching values are stored on the respective node. To avoid redundancy, each request is forwarded only once by each node. In contrast to the MADPastry-based approach, neither bootstrap nor insertion phases are necessary as nodes store their values locally and do not need to distribute them.

The results of these simulations are illustrated in Figures 2 and 3. In Figure 2, the recall is plotted against the number of values provided by each node. We can observe that DHT-based range queries using MADPastry achieve recalls between ca. 90% – 95% for up to 5 values per node for both one- and two-byte values. For 10 values per node, the one-byte values can still maintain a recall of around 90% whereas the two-byte values' recall drops to around 70%. For even higher numbers of values per node, the recall degrades significantly. The reason for this is that the larger the number of values, the longer the handover packets (see Section 2) become that are necessary due to node mobility. This, in turn, increases the probability of packet collisions, specifically with query and reply packets.

Generally speaking, range queries using MADPastry perform considerably better for smaller value ranges. This is due to the fact that, with increasing value ranges, the depth of the DST, and, thus, the total number of DST nodes increases. Hence, each network node will be responsible for an increased number of DST nodes, which, in turn, increases the amount of handover packets caused by node mobility. In contrast, broadcast-based range queries are only able to retrieve 70% of the values available in the network in the simplest scenario where only a single value is provided by each node, and between 45% and 50% of the values for higher numbers of values per node.

Additionally, MADPastry's recall for 15 and 20 two-byte values per node deteriorates drastically, as can be seen in Figure 2. Note that the main reason for this unusually steep decline is the fact that the initial 10-minute insertion period is actually not sufficient for these scenarios. Due to the sheer amount of necessary insert packets and frequent collisions with other packets, up to the very end of the simulated hour, large numbers of inserts had not been successfully delivered. Thus, large proportions of expected values could not be retrieved, which significantly lowers the recall.

When looking at the network traffic incurred by range queries and replies as illustrated in Figure 3, one can see that, for up to 5 values per node, MADPastry achieves it much better recalls by generating comparable or even lower network traffic than the broadcast-based approach does. Furthermore, while the increase in network traffic in the case of simple broadcasting is negligible, it is linear in the number of values per node in the case of MADPastry. This is, again, due to the growing number and length of handover packets.

4. CONCLUSION

The primary goal of this paper was to gain first insights into the question whether range queries in MANETs can be implemented using a DHT. For this purpose, we have adapted a DHT-based range query approach employing a Distributed Segment Tree proposed for the Internet domain [4] to the use in MANETs on top of the DHT substrate MADPastry [3].

First experimental results show that this approach can support efficient range queries in MANETs given reasonable bounds in terms of the number of values provided by each node as well as the range of possible values. For large numbers of values and value ranges, the performance suffers from the necessary maintenance of both the DHT and the range query data structure (DST). Therefore, the applicability of our approach depends largely on the application scenario. For example, in a wireless sensor network, it is reasonable to assume that only a very limited number of values will be provided by each sensor node and only a limited range of possible values (e.g. temperature readings) can be expected. Moreover, efficient DHT-based range queries might be generally interesting in MANETs that are already running a DHT-substrate for application purposes in the first place.

In the future, DHT-based range querying needs to be more thoroughly evaluated with respect to different network scenarios. Specifically, one should look at parameters such as network size, node density, mobility, and query patterns to name but a few. Of particular interest to our approach might also be the ratio between physical nodes and the number of nodes in the DST.

On a more general note, we have identified handover packets caused by MADPastry cluster changes due to node mobility to be a major traffic source that adversely affects performance. Therefore, we believe further research is needed on the trade-off between shorter physical paths (as provided by MADPastry's clusters compared to non-locality aware DHT substrates) and additional handover traffic (which is not incurred by non-locality aware DHT substrates). On the other hand, it might be even more interesting to try and leverage MADPastry's clustering feature for a more efficient mapping of the DST nodes to network nodes.

5. REFERENCES

[1] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, and A. Rowstron. Virtual Ring Routing: Network routing inspired by DHTs. To appear at *ACM SIGCOMM'06*, Sep. 2006.

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

[3] T. Zahn and J. Schiller. MADPastry: A DHT Substrate for Practicably Sized MANETs. In *Proc. of ASWN 2005*, Jun. 2005.

[4] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed Segment Tree: Support of Range Query and Cover Query over DHT. In *Proc. of IPTPS'06*, February 2006.