A Hierarchical Approach to Resource Awareness in DHTs for Mobile Data Management

Liz Ribe-Baumann
Ilmenau University of Technology
Ilmenau, Germany
liz.ribe-baumann@tu-ilmenau.de

Kai-Uwe Sattler
Ilmenau University of Technology
Ilmenau, Germany
kus@tu-ilmenau.de

ABSTRACT

Data is increasingly distributed across networks of mobile nodes such as wireless sensor networks, distributed smartphone applications, or ad hoc recovery networks in disaster scenarios, but must still be reliably collected, stored, and retrieved. While such networks run in either ad hoc mode or use existing infrastructure, all of them must deal with node heterogeneity. Wireless nodes invariably have differing levels of power availability, and often varying connectivity and computing power. While many distributed hash tables (DHTs) have been designed for mobile ad hoc or heterogeneous networks, they do not consider differences in node strength, or *resource availability*, for an arbitrary number of resource availability levels. In this paper, we present a scalable, location aware, hierarchical DHT that utilizes nodes' varying resource availability levels to increase and prolong the mobile network's data storage and retrieval capabilities. Furthermore, we compare this DHT to other location aware flat and hierarchical approaches, examining their structures' suitability for nodes with varying resource availability.

1. INTRODUCTION

Today, mobile applications are no longer restricted to the classic client/server architecture relying on a backbone infrastructure. Instead, an increasing number of applications for smartphones (e.g. mobile games, content sharing) as well as wireless sensor network applications follow a serverless ad-hoc model of interaction and data exchange. Even if such applications require a server or gateway to initially fetch some data or to finally publish results, data has to be collected, exchanged, and stored for some time in the network. From a data management point of view, this poses two challenges: (1) to *efficiently* manage and retrieve data in a distributed way and (2) to *reliably* provide the data while taking possible node failures and resource restrictions (connectivity, battery power) into account.

Distributed hash tables (DHT) for mobile peer-to-peer (P2P) networks have been proposed in the past to address the first challenge. However, constructing a distributed hash table on a mobile P2P network, where nodes have restricted battery power and communication costs the peers vital energy, is a definite challenge. While traditional DHTs such as the Content Addressable Network (CAN) [24] and Chord [29] may be reliable on a network without such communication restraints, they fail to take the important differences in nodes' resource availabilities into account for resource sensitive dynamic networks. And while today's large mobile networks are based on smartphones, laptops, etc. which use an intact backbone infrastructure that nodes need not consider, the use of DHTs on ad-hoc networks should be considered for the near future.

A fundamental operation for any kind of data management task (store, update, retrieve) in a DHT is the key lookup operation. To implement this operation in a (mobile) ad-hoc network, overlay nodes also forward messages on the network layer, in which case a long distance overlay hop may require many forwarding nodes, while a short distance lookup hop may be completed with few network hops. This case brings the additional challenge of routing distances, giving the physical distance that lookups traverse a central role in a network's ability to survive its load. Thus, power and (in the future) location awareness are important for DHTs running on mobile networks.

Consider for example a large network based on smartphones, laptops and a limited number of servers that cooperatively maintain a DHT, with each node storing some of the global application data. As long as the load is balanced between the nodes, this network is inherently scalable - each of the nodes is responsible for fetching and storing a portion of the network's information and for routing and processing a portion of the network's requests. While the numerous smartphones jointly provide a large portion of the network's storage and routing capabilities, a single smartphone has a restricted amount of battery power available in a give time frame (until it is recharged). Thus, the network must find a way to incorporate each of these weaker nodes' resources in order to provide scalability without causing failure by overuse. In this paper, we address the second of the above mentioned challenges by examining several approaches to balancing maintenance and routing load according to node's resource (i.e. power) availabilities and locations. None of these approaches acts blindly with regard to either resources or locations, yet each has clear limitation with regard to how much it can incorporate. Since we could, for example, consider a node's restricted computing power or bandwidth availability instead of its power availability, we consider the

abstract notion of a node's "resource availability."

The three major approaches towards addressing heterogeneous node capabilities in DHTs - hierarchical DHT structures, virtual nodes, and node movements within the identifier space - are typically not well adept to our scenario. Hierarchical approaches, typically with two tiers [3, 13, 36], offer large reductions in the load on leaf nodes but overuse peers with restricted resources which act as super peers (in order to assure the system's scalability). On the other hand, virtual nodes and node movement approaches, which vary the quantity of data at each physical node, actually introduce more maintenance overhead and churn while assuming that higher resource availability implies larger storage capacity. However, since the main communication overhead in a DHT comes from maintenance, we are most interested in balancing the network maintenance and lookup routing load to nodes' resource availabilities.

We compare four different structural resource and location aware DHT approaches in this paper: (a) a novel, multi-tiered hierarchical approach, (b) a two-tiered hierarchical approach, (c) a flat resource and location aware approach [27], and (d) a (novel) hybrid approach between (a) and (c). While all three hierarchical approaches treat the lowest level nodes as leaf nodes, approach (a) constructs multiple upper tiers, approach (b) uses location aware DHash++ [9] for the upper tier, and approach (d) uses the location and resource aware Chord extension RBFM [27] for the upper tier. We examine each network's ability to store and retrieve data, as influenced by the nodes' lifetimes and the percentage of deliverable lookups. This paper's main contributions are:

- A novel location aware hierarchical DHT for an arbitrary number of resource availability levels and
- a simulated comparison of network robustness for four flat and hierarchical resource and location aware approaches.

We discuss related work in Section 2; explain our network assumptions and foundations in Section 3; describe our novel DHTs and consider the routing complexity of the approaches in Section 4; and compare the four DHTs using simulation in Section 5.

2. RELATED WORK

The DHT forerunners such as CAN (Content Addressable Network) [24], Chord [29], and Kademlia [21] use efficient routing but were not designed to run on mobile nodes where both location and available resources play important roles. Proximity-awareness in DHTs is generally classified as proximity-aware identifier selection (PIS, such as Mithos [31] and SAT-Match [26]), proximity-aware neighbor selection (PNS, such as DHash++ [9]), proximity-aware route selection (PRS, such as Tapestry [35]), or a combination thereof and is primarily directed at reducing overall traffic or average round trip times [16]. Proximity-awareness has gained interest in many areas related to DHTs, including caching and replication protocols and hybrid overlays [10, 20].

The three main approaches for balancing load in heterogeneous DHTs are the use of hierarchical DHT structures, virtual nodes, and node movements within the identifier space. Hierarchical DHTs often group nodes by some defining characteristic such as group associations (e.g. departments within a university) or peer capabilities ("have" or "have not"). Systems with group structures such as Canon [12], Hieras [32], and Cyclone [2] tend to route lookups as far as possible within one group before forwarding them on to a different (often hierarchically higher) group. In contrast, hierarchical DHTs based on two-tier peer capabilities [3, 36], where nodes assume the roles of super-peer or leaf-peer, route lookups directly from leaf nodes to parent nodes, rendering the parent nodes fully responsible for performing lookup routing and neglecting the varying nuances of nodes' resource availabilities. A combination of the group and two-tiered capabilities structures has also been suggested in [13], such that weak peers are arranged in disparate DHTs controlled by super-peers which form their own DHT and are responsible for routing lookups to the correct group.

Virtual nodes pose a different solution, with each physical network node balancing its load independently by hosting a varying number of virtual overlay nodes, each with its own set of keys and links [15, 18]. Similar to virtual nodes, node movements within the identifier space achieve load balance by adjusting the data that each node stores [5, 11]. Nodes with low load choose new nodeIDs that are close to nodes with high load, thus taking over some of their load but creating a large amount of churn.

DHTs for mobile ad hoc networks (MANETs) pose additional challenges since each overlay hop represents multiple underlay hops on DHT nodes, underlay routing to unknown destinations often results in broadcast messages, nodes' mobility causes frequent changes in "good" routes, and high node churn due to short uptimes (and dwindling resources) requires highly dynamic protocols for overlay maintenance and data persistence. Among the first DHTs suggested for MANETs were Ekta [23] which uses underlay routing information to choose links (PNS) and make overlay routing decisions (PRS), MADPatry [34] which uses landmark nodes to form location-based node clusters that share nodeID prefixes (PIS with node movements), and CHR [1] in which geographic clusters act as nodes using geographic routing as in GHT [25]. Basically, DHTs in MANETs employ some combination of cross-layer PIS [22, 14, 30, 33], PRS [19], and PNS [7], using network layer (i.e. underlay) information to augment overlay decisions. Many of these overlays can be considered hierarchical [22, 30, 33, 34], in part due to their clustered structures.

While proximity-awareness plays a central role in the development of DHTs for wireless networks, the treatment of nodes' heterogeneity also effects the robustness of the final system, especially when considering nodes' power availability and connectivity. The numerous mentioned DHT substrates handle node heterogeneity differently - ignoring it completely or incorporating it in a flat or hierarchical manner - but there exists limited work comparing these various approaches. Furthermore, little has been done to treat nodes with varying resource availabilities in DHTs with mobile nodes on a finer scale than strong or week, as in the Chord extension RBFM [27].

3. MOBILE NODE RESOURCE MODEL

Coordinates. We assume that each node x has sufficiently correct two dimensional virtual (not necessarily geographical) network coordinates (x_1, x_2), such as used in Vivaldi [8] for determining latencies. The *physical distance* $d_{phy}(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 + y_2)^2}$ between two nodes x and y and should reflect round trip times, the number of

underlay hops, or some other meaningful distance between nodes.

Resources. We model our analysis and simulations after nodes with varying power availability - from smartphones with very limited power to servers with an inexhaustible power source - but the proposed protocol need not be restricted to this use case. It only requires that each node x has a resource availability that can be expressed as an integer value $x_R \in \{0, 1, \ldots, l_{max}\}$ for some fixed maximum level l_{max}. We assume that $x_R = 0$ is the lowest possible resource level (but still operational) while $x_R = l_{max}$ implies unbounded resources. Note that resource levels must be globally defined so that a given resource level on differing node types is comparable. If we consider power availability with $l_{max} = 3$, that could mean that a handheld operating on battery power may have resource level two when fully charged, but a cell phone with a weaker battery may only reach a resource level one when fully charged.

For our simulation, we use a Zipf distribution for nodes' resource levels, reflecting trends for node lifetime found in peer-to-peer networks, where node lifetime tends to follow a heavy-tailed Pareto distribution [6, 28] (the continuous counterpart of the Zipf distribution). The probability that a random node has resource level $\ell \in \{0, 1, \ldots, l_{max}\}$ depends on the power m of the Zipf-distribution:

$$p_\ell := P(x_R = \ell) = \frac{1}{(\ell+1)^m} \cdot \frac{1}{\sum_{j=0}^{l_{max}} 1/(j+1)^m}. \quad (1)$$

Failures. For our simulation, we assume that failures are due to nodes' resources being depleted by node activity. Each send and receive activity drains a node's resources until the node fails (based on nodes with varying power availability). We use asymmetrical drain patterns, with a send costing more than a receive, but constant drain for all but the top level, which is not drained at all. The selected resource and drain values are abstract and serve as a benchmark to compare the protocols as opposed to assessing the real word battery runtimes. To provide results with as few dependencies as possible, we take a simplistic approach to churn in our evaluation, with nodes drained until they fail but no additional nodes added. The system runs until it has been reduced to half of the original nodes.

DHT Foundation. All of the approaches we use are based on Chord [29] mainly because Chord has a rather simplistic structure that adapts well to location awareness [16] and is the basis of the location aware DHash++. Analogous to Chord, we use consistent hashing [17] to distribute keys to nodes. Each node x chooses a random (or hashed) nodeID x_{ID} from the binary key space $0 \ldots 2^m - 1$, which is viewed as a ring with key values increasing in a clockwise direction. These completely random nodeIDs ensure a scalable key distribution. Each node positions itself at its nodeID on the key ring and establishes links to its immediate predecessor and successor as well as a successor list with its r nearest successors, making repairs possible after unexpected node failures. Each node x maintains a *simple key range* $x.srange$, which spans the keys between its predecessor y's key (exclusive) and its own key, or $x.srange = (y_{ID}, x_{ID}]$. Thus, each key κ is assigned to the first node whose nodeID is equal to or succeeds κ on the key ring, or that node whose simple key range contains κ. The asymmetric *key distance* from a node x (or key) to a node y (or key) via their nodeIDs is:

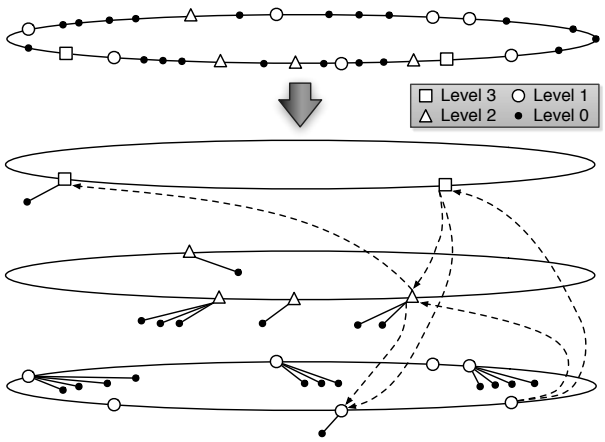

Figure 1: All resource levels shown on top key ring. Nodes within hierarchical layers below: upper layers form DHash++ overlays, links between layers, and leaf nodes assigned to upper level predecessors.

Key Distance 1. The key distance from x to y is the clockwise distance on the key ring from x_{ID} to y_{ID}: $d_{key}(x, y) := y_{ID} - x_{ID} \mod 2^m$.

4. HIERARCHICAL OVERLAY

We call our hierarchical approach Hierarchical Resource Management (HRM), where nodes are separated into different levels and maintain links within and between those levels. The lowest level, consisting of the weakest nodes, functions as a leaf level where each leaf node maintains a parent node from some upper level. Each level is linked together to form a location aware DHash++ [9], with the number of shortcut links determined by the given hierarchy level. Thus, the higher a node's resource availability, the more links it is expected to maintain, so that weaker nodes' maintenance loads are significantly reduced despite their additional load as parent nodes. We assume that all nodes play similarly important roles in data storage and retrieval, thus, we do not address heterogeneous data distribution or the necessary replication protocols in this paper.

In the following we refer to *bottom level* nodes with resource level $= 0$, *upper level* nodes with resource level > 0, *top level* nodes with resource level $= l_{max}$, and lower level nodes with resource level $< l_{max}$. In addition to each node's links to its predecessor and first r successors, we have three additional types of links: leaf-parent links between bottom and upper level nodes; inter-level links which connect each upper level node with its immediate successor in each of the $l_{max} - 1$ upper levels; and level fingers which provide each upper level node shortcuts within its own resource level.

4.1 Links.

Each node is responsible for the keys in its simple key range, but we also consider an upper level node's *upper key range* which contains all of its leaf nodes' nodeIDs and is integral to successful routing. For this, each node maintains upper level successor and predecessor nodes, i.e. the first successor and predecessor nodes from any upper level. Then a node x's *upper key range* $x.urange$ consists of all keys between x_{ID} and its upper level successor's key. Note that $x.srange$ and $x.urange$ overlap only in x_{ID}.

ProspectiveLinks[x_R][i]

NodeID	Physical Distance
p_2ID	0.4
p_3ID	0.9
p_4ID	0.9
p_1ID	1.7

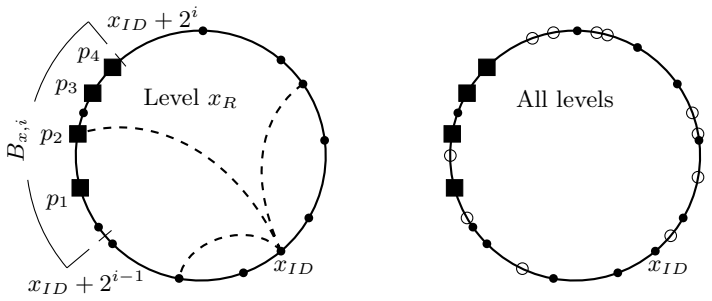

Figure 2: Key ring at right shown for node x with all nodes not in level x_R as hollow circles and at left with level x_R nodes only: six nodes in $B_{x,i}$, four of which x knows in its x_R prospective links list (squares). A level finger is established to p_2, the known node with the best physical distance to x in level x_R.

Leaf Links. Each bottom level node x ($x_r = 0$) maintains a link to its *parent node* $\pi(x)$, which is the first upper node preceding x in the keyspace. Thus, leaf nodes have parents from varying resource levels. Leaf nodes have neither inter-level links nor level fingers.

Inter-level Links. Each upper level node x establishes a link $x.I[\ell]$ to its direct successor $x.I[\ell].node$ in each of the upper levels ℓ.

Level Fingers. In DHash++, each node x with nodeID x_{ID} chooses one link - or finger - $x.F[i]$ per finger interval $B_{x,i} := [x_{ID}+2^{i-1}, x_{ID}+2^i)$ for $i \in \{1, 2, \ldots, m\}$. The corresponding node that $x.F[i]$ points to is notated $x.F[i].node$. In our protocol, a node x only choses fingers within the same resource level, i.e. $(x.F[i].node)_R = x_R$. Furthermore, the number of fingers that a lower level node establishes varies from level to level.

A level 1 node x has as few fingers as necessary, establishing level fingers only to nodes which are closer successors in the keyspace than x's closest higher level inter-level link. That means:

$$d_{key}(x, x.F[i].node) < d_{key}(x, x.I.closestHigher). \quad (2)$$

Where $x.I.closestHigher$ is the closest of x's higher level inter-level links, $x.I.closestLevel$ is $x.I.closestHigher$'s resource level, and $x.I.closestInt$ is the finger interval in which $x.I.closestHigher$ is found:

$$x.I.closestLevel := \underset{\ell: x_R < \ell \le l_{max}}{\text{argmax}} \; d_{key}(x, x.I[\ell].node)$$

$$x.I.closestHigher := x.I[x.I.closestLevel].node$$

$$x.I.closestInt := j : x.I.closestHigher \in B_{x,j}.$$

Meanwhile, level l_{max} and $l_{max}-1$ nodes maintain fingers for each finger interval $B_{x,i} := [x_{ID} + 2^{i-1}, x_{ID} + 2^i)$ for $i \in \{1, 2, \ldots, m\}$. Nodes in additional levels ℓ with $1 < \ell < l_{max} - 1$ maintain sets of fingers of varying sizes, depending on ℓ. We let $x.Finterval \in \{1, 2, \ldots, m\}$ be the furthest finger interval in which a node x maintains a finger and $x.Fkey = x_{ID} + 2^{Finterval-1}$ its corresponding key value. For example, given five levels ($l_{max} = 4$), we might have:

$$x.Finterval = \begin{cases} x.I.closestInt, \; x_R = 1 \\ m-1, \; x_R = 2 \\ m, \; x_R \in \{3, 4\}. \end{cases}$$

Thus, each upper level node x maintains a *finger table* with (at most) one finger for each finger interval $B_{x,i}$ with $i \in \{1, 2, \ldots, x.Finterval\}$. Note that the fewer links a level maintains, the less maintenance load is incurred and the faster messages are passed on to other (higher) levels. Lookups are thus routed quickly out of the bottom layers and dispersed between the upper layers.

Level fingers are chosen in a location aware fashion as in DHash++. Nodes' coordinates and resource levels are piggybacked on network messages, providing node information to other nodes at minimal overhead. Thus, an upper level node x choses for $x.F[i]$ that known node with resource level x_R in the finger interval $B_{x,i}$ which has the smallest physical distance to x. For this, x maintains a set of l_{max} *prospective links* lists, one for each resource level, with the ℓ^{th} prospective links list containing a list of the closest (in terms of physical distance) k nodes in $B_{x,i}$ for each $i \in \{1, 2, \ldots, m\}$ from level ℓ which are known to x. At most k nodes in $B_{x,i}$ are saved via their resource levels, nodeIDs, and physical distances, so we have at most $k \cdot m \cdot l_{max}$ saved nodes.

When x receives a message that originated at sender y, x uses y's coordinates to determine $d_{phy}(x, y)$ and update its level y_R prospective links list accordingly (see Figure 3). An i^{th}-finger request is sent to the closest entry in x's level x_R prospective links list for $B_{x,i}$, if it contains an entry (see Figure 2). Otherwise, the first successor of key $x_{ID}+2^i-1$ in resource level x_R is contacted (see Figure 3), which requires level-specific lookup forwarding (see Routing). Upon node x's receipt of a finger request response from node y, if $y_R = x_R$ and $y \in B_{x,i}$ with $i \le x.Finterval$, then y is assigned to $x.F[i]$. If a given finger interval contains no level x_R node, then this finger entry remains empty.

Note that while node x only links level fingers to nodes in level x_R, maintaining prospective links lists for multiple levels causes little overhead while easing a node's transition between resource levels. The prospective links list entries are continually updated with fresh node information to automatically adapt the network to changing coordinates and are deleted once used for a finger request to ensure their freshness. Simulations have shown that $k = 1$ is beneficial in networks with high churn, reducing the use of failed prospective links and minimizing the lists' overhead.

Figure 1 shows the basic overlay stucture. The connected key ring on which each node establishes its predecessor and successor is shown on top, and the individual levels are shown below with the bottom level nodes assigned to their upper level predecessors (i.e. parents). Inter-level links are shown for three nodes only and level fingers were omitted.

4.2 Node Joins and Failures.

To join the DHT, a node x must have valid network coordinates, choose a nodeID and resource level, and contact

```
procedure MAINTAINFINGER(finger)
    lookupKey = myKey + getOffset(finger)
    myLevelList = prospLinkList(myLevel)
    if myLevelList.size(finger) > 0 thenlevel
        listEntry = myList.getClosestEntry(finger)
        lookupKey = listEntry.key
        myList.removeUsedEntry(listEntry)
    end if
    sendLookup(lookupKey)
end procedure
```

```
procedure SUGGESTPROSPECTIVELINK(nodeInfo)
    finger = getFingerInterval(nodeInfo.key)
    dist = getPhysicalDist(nodeInfo.coordinates)
    level = nodeInfo.resourceLevel
    if pLList(level).contains(finger, nodeInfo.key) then
        pLList(level).update(finger, dist, nodeInfo)
    else if pLList(level).size(finger) < k or dist <
pLList(level).farthestLinkDist(finger) then
        pLList(level).addNode(finger, dist, nodeInfo)
        while pLList(level).size(finger) > k do
            pLList(level).removeFarthestLink(finger)
        end while
    end if
end procedure
```

Figure 3: Maintaining fingers 1 to $m-1$; Updating prospective links lists (pLList) with $\leq k$ entries

one participating node. Once x has established links to its immediate predecessor p and successor s on the key ring, s sends its successor lists to x, which x uses to initialize its own list, and corresponding keys are transfered from s to x. Once x has completed the basic join in the overlay, it must also perform either a leaf join or upper level join (see below). The node x continually updates its *prospective links* lists and periodically performs finger maintenance (see Figure 3) to establish and maintain its fingers.

The basic reaction to node failures is as in Chord, with failed nodes also removed from the inter-level list, prospective link lists, and potentially the parent link or leaf list once their failure is noticed. An upper level node leaves gracefully by sending messages to each of its leaf nodes and its upper level predecessor informing them of their new parents/leaf nodes. Otherwise, if a leaf node's parent has fails unexpectedly, the leaf node must perform a leaf join to reestablish a parent (see below).

Upper Level Joins and Failures. The upper level join serves two purposes: establishing the upper level successor and predecessor nodes (from any upper level) and transferring the responsibility for leaf nodes. Node x uses an upper level bootstrap node to send an upper level join message, which is routed along the upper level nodes to x's upper level predecessor y, for which $x_{ID} \in y.urange$. Node y responds to x with its own upper level successor z and the list of y's leaf nodes which are now in $x.urange$. Then y informs each of these leaf nodes of their new parent node x and removes them from its leaf list.

If an upper level node's resource level is reduced to the lowest level, it becomes a leaf node and forfeits its role as parent node by transferring its leaf nodes to its upper level predecessor y. Leaf nodes ignore finger and inter-link requests, upper and leaf join requests, and upper stabilize requests. If it is observed that a node has left the upper levels, it must be removed from inter-level list, prospective link lists, upper level successor and predecessor links, and parent links (but not from successor and predecessor links).

Leaf Joins and Failures. Nodes with resource level 0 perform leaf joins to establish a live parent node. Recall that a node x's parent is the first preceding upper level node on the key ring. The message is forwarded to an upper level bootstrap node and then routed to the upper level node whose upper level key range contains x_{ID}. This parent node responds and and enters x into its leaf list.

Maintenance. Given the dynamics of mobile networks, maintenance is integral for detecting and addressing network changes. Inter-level links and level fingers are maintained similar to in RBFM with varying maintenance intervals and are automatically adapted when nodes change resource levels. Thus, each link is maintained at an interval that depend on the link's node's resource level: Bottom level links are maintained according to a reference interval t_{ref} and higher level links at varying multiples of t_{ref} for each resource level.

However, leaf and parent links as well as upper level successor and predecessor links are maintained analogously to direct successor and predecessor links, using direct maintenance messages at a fixed interval.

4.3 Lookup Routing.

Routing of lookups is not performed in a strictly greedy fashion like Chord, but rather in a series of greedy steps. Let κ be the message's destination key. Now recall that a message is destined for the node whose simple key range contains κ. One negligible piece of information is added to messages: the key of the last upper level node that handled the message (only needed for high churn scenarios). Once a node x has determined that $\kappa \notin x.srange$, its routing behavior depends on its resource level as follows:

Leaf Nodes. If a message originates at a leaf node x, it is forwarded to the parent node. Otherwise, if x receives a messages for which its parent node was not the last upper level node to have handled the message, it forwards the message to its parent node. Otherwise, it forwards the message to its successor node y if $\kappa \in y.srange$ or else to the closest preceding node from its successor list.

Upper Level Nodes. An upper level node x routes greedily to the closest preceding node in a resource level $\geq x_R$ using both its finger table and inter-level list. For lower level nodes with restricted finger tables, this means that messages to 'distant' destinations are routed to upward. For top level nodes, this means that messages are routed to the closest top level predecessor of κ.

If there is no closer predecessor node with resource level $\geq x_R$, then the message is routed back down the hierarchy by choosing the highest possible inter-level link preceding κ. If there are no such links, then $\kappa \in x.urange$ and the message is delivered directly to the node y with $\kappa \in y.srange$: y is either x's upper level successor (whose simple key range overlaps $x.srange$) or one of x's leaf nodes.

Level Routing. Finger requests and inter-level link requests use level sensitive overlay routing. These requests are not necessarily delivered to the node x for which $\kappa \in$

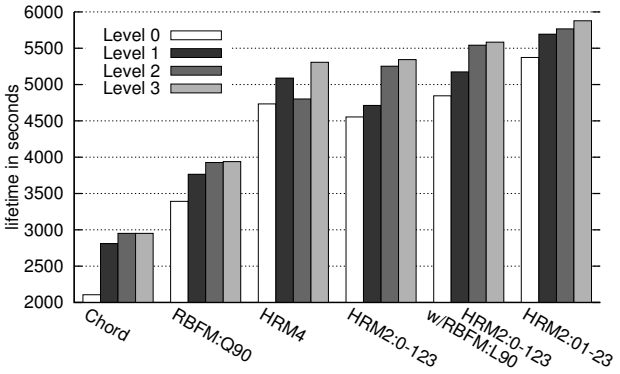

Figure 4: Mean node lifetimes per resource level until half of the system nodes have failed due to resource drain. Level 3 is also total system lifetime.

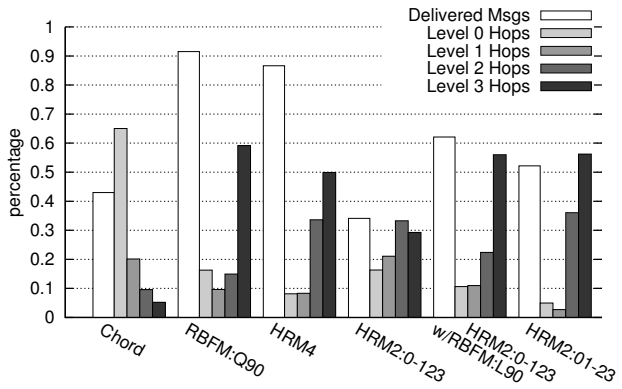

Figure 5: With node failures: The percentage of delivered application messages and percentage of total lookup hops resolved per resource level.

$x.srange$, but rather to the first successor of κ in a given level ℓ. A node y considers itself the request's destination if $y_R = \ell$ and κ is between the sending node's nodeID and y_{ID}. If not, y forwards to its level ℓ inter-level link if it succeeds κ, or to its closest known preceding link in level ℓ.

Lookup Hop Length Theorem 1. *Given a network with N nodes, the expected upper bound for the number of overlay hops required for a lookup from any node to the successor node of any key is $O(\log(N))$ hops.*

Proof. To show that routing terminates and find an upper bound on the routing hops, we consider the farthest possible lookup route. Since it takes at most one hop to route from a leaf to a parent node, we assume that each message originates at an upper level node. Furthermore, since it takes at most one hop to reach the node y with $\kappa \in y.srange$ from κ's upper level predecessor, we need only determine the number of hops necessary to reach κ's upper level predecessor.

If the originating node, its successor, or its upper level successor are the destination, then we are done. Otherwise, the message is passed upwards until it reaches the first level in which the destination key is a predecessor of the current node x's $x.Fkey$, i.e. $d_{key}(x,\kappa) < d_{key}(x, x.Fkey)$. The message is passed up at most $l_{max}-1$ levels, from the bottom to the top level.

So assume that κ is a predecessor of $x.Fkey$ and will thus be routed within level $\lambda := x_R$ on level fingers until it has reached either the destination node or the closest predecessor node within level x_R. Then, as shown in [27], the routing complexity within level x_R is at most $O(\log(x.N))$, where $x.N$ is the number of nodes in level x_R between x and $x.Fkey$. Let y be κ's closest predecessor node in this level.

Assuming the message has not reached its destination, the message is passed at most once to each of the other upper levels and routed analogously (starting at higher levels first). So we have a total of $l_{max}-1$ remaining hops between other upper levels. However, since we know that y is the closest predecessor node to κ in level λ, we can use the expected number of network nodes between any two level λ nodes (some constant c) as an upper bound on the number of nodes on which remain to route over in each level. Thus, we expect at most $O(\log(c))$ routing hops in each remaining level $\ell > 0, \ell \neq \lambda$, giving us an expected total of at most:

$$\begin{aligned} 2 + 2(l_{max}-1) &+ O(\log(x.N)) + (l_{max}-2)O(\log(c)) \\ &\leq 2 \cdot l_{max} + \hat{c} + O(\log(N)) \\ &= O(\log(N)) \end{aligned}$$

for with some constant $\hat{c}$. □

Note that our simulation results did show an increase in routing hops compared with Chord (1-2 hops), as expected due to the maximum $2 \cdot l_{max}$ hops up and down the hierarchy.

5. EVALUATION

In order to assess the various approaches' capabilities to store and retrieve data, we observe node and network lifetimes, the ratio of successfully delivered lookups, and the ratio of forwarded lookup hops per resource level. Additionally, as a reflection of each approach's suitability to mobile scenarios, we compare the average physical distance of lookup routing hops. We use two simulation configurations, one without node failures and one in which nodes' resources are drained upon message send and receives, terminating the simulation once half of the nodes have failed (as described in Section 3). We consider here only systems with four resource availability levels ($l_{max} = 3$), leaving considerations about the ideal number of resource levels for future work. We compared the HRM overlay with Chord, RBFM, and three two-tier hierarchical overlays based on Chord, RBFM, and HRM. We found the variations caused by different RBFM finger maintenance intervals and stretch constants insignificant compared to the variations between approaches, so we chose to use fixed RBFM configurations.

5.1 Setup

The simulations were performed in OmNET++ using the OverSim overlay framework [4], using and extending the functionality of the existing Chord implementation. The results are based on 10000 nodes with random coordinates divided into four resource availability levels based on the power two Zipf distribution from (3). The base measurement time was 10000 seconds, but the simulations with node failures were terminated once half of the nodes failed. Nodes started with resource values 800, 200, and 100 for levels 2, 1, and 0 (level 3 nodes are not drained due to their inexhaustible resources) and drained by 0.2 and 0.1 resource

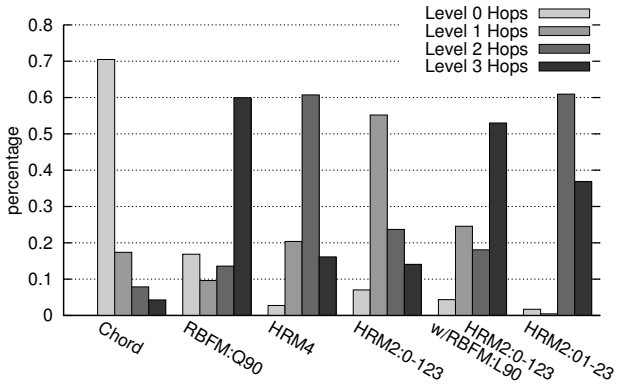

Figure 6: Without node failures: Percentage of total lookup hops resolved per resource level. Each approach delivered more than 99% of lookups.

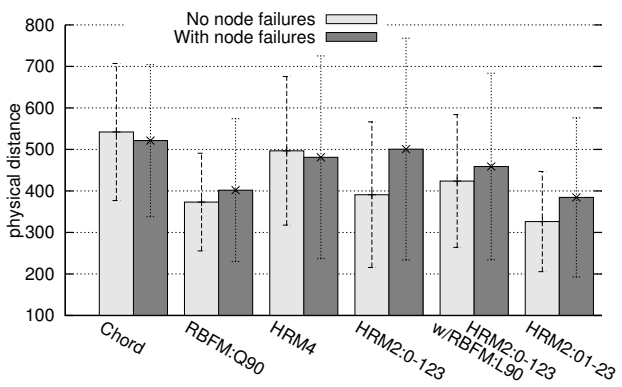

Figure 7: Mean physical distance and standard deviation per lookup routing hop.

units for every sent and received message, respectively. In addition to maintenance messages (including stabilize messages every 20 seconds to node successors and parent nodes), dummy application lookups were sent from each node to a random key every 30 seconds. The application lookups were observed for the number of hops, hop distances, and successful delivery. The lower the rate of delivery, the less reliable the DHT stores and retrieves data.

5.2 Hybrid hierarchical DHT

The three different two-tier configurations we simulated in order to compare HRM with classical two-tier approaches consist of parent and leaf nodes. Nodes from each of the four resource availability levels are assigned to either a super peer or a leaf peer hierarchical layer. For HRM2:0-123, level 0 nodes are assigned to the leaf layer and level 1,2, and 3 nodes to the super peer layer. Within the super peer layer, nodes are unaware of their varying resource levels and choose their fingers based only on physical distance. Similarly, HRM2:01-23 assigns level 0 and 1 nodes to the leaf layer and level 2 and 3 nodes to the super peer layer.

In order to add additional resource awareness, we combine HRM with RBFM to obtain a hybrid solution: in HRM2:0-123 with RBFM nodes are again arranged with level 0 nodes in the leaf layer and level 1,2, and 3 nodes in the super peer layer. However, here the super nodes are aware of their different resource levels and choose fingers in a resource and location aware fashion as in RBFM. Thus, the upper level nodes build a simple RBFM overlay on which the bottom level nodes are hung as leaf nodes. The two-tier simulations were configured with linear finger maintenance and for HRM2:0-123 with RBFM using the stretch constant $c = 90$.

5.3 Results

Further simulation configurations include: a relatively infrequent finger maintenance period of 120 seconds for Chord for reduced load, an RBFM overlay with quadratic finger maintenance and stretch constant 90 (RBFM:Q90) (see [27]), and HRM as suggested in this paper with four hierarchical levels and linear finger maintenance (HRM4) with finger intervals (as described in Section 4):

$$x.Finterval = \begin{cases} x.I.closestInt, \; x_R = 1 \\ m, \; x_R \in \{2, 3\}. \end{cases}$$

The lifetimes of each of the systems before half of its nodes fail, as shown in Figure 4, vary strongly. Despite a very infrequent finger maintenance interval for Chord, set here intentionally to 240 seconds to reduce maintenance load, it cannot compare to the alternative approaches. Note that these results depend on the varying maintenance loads which we do not further discuss for the sake of brevity, but which produce a major portion of network load. Note that while HRM4 performs significantly better than RBFM, the two-tier approaches perform even better, most likely due to the higher average hop length of lookups in HRM (approximately one hop more per lookup). While Figure 4 demonstrates how well the traditional two tier approaches prolong node lifetimes, Figure 5 shows that these two-tier approaches' performance suffers given high failure rates, reducing the success rate of lookups to under 65%. Considering only the node and network lifetimes together with the percentage of delivered lookups, RBFM and HRM4 clearly outperform the other approaches, with HRM4 providing the longer lifetimes.

Figures 5 and 6 also provide an overview of the lookup hop load distribution among the resource levels with and without node failures. Only successful lookups are included in these figures. Figure 6 clearly shows how strongly each approach prefers a single resource level for its lookup hops and demonstrates how important it is to design an overlay with the nodes' resource availabilities in mind. For example, RBFM's lookup hop distribution is more suitable for networks in which top level nodes can handle unlimited load while HRM4 is more suitable for systems with strong nodes in level 2 that should be used to reduce top level load.

Average lookup lengths ranged from 6.5 to 8.5 hops, with the hierarchical approaches tending to have around one hop more than the other approaches, presumably from the final and/or initial hops to and from leaf nodes. Note the high load on level 1 nodes for the two-tier approach HRM2:0-123 in Figure 6. We infer that this causes a high rate of node movement between the super peer and leaf layers, triggering HRM2:0-123's low deliverabiliy rate. Similarly, HRM4 distributes more load to level 2 nodes at the cost of their average lifetimes (relative to the other levels) as seen in Figure 4.

The average physical distance of single lookup hops shown in Figure 7 reflects what we expect of the approaches' routing hops' distances: while RBFM:Q90 has a large pool of nodes from which to choose links with strong and physically close nodes, HRM4's nodes have less choice for their links

which are drawn from specific (less populated) hierarchical layers. HRM4 actually performs only slightly more location-aware than the completely location-naive Chord - reducing its usability for mobile network scenarios with ad hoc routing where location-awareness is integral: multi-tiered hierarchical approaches for MANETs require additional location-awareness such as PIS or PRS. On the other hand, level 2 and 3 nodes in HRM2:01-23 choose links from these two upper levels based only on distance, providing upper nodes with physically close links. Thus, depending on the tolerable lookup failure rate and desired distribution of load between the upper level nodes, HRM2:01-23 and RBFM are best suited for systems where the physical routing distance plays a central role.

6. FUTURE WORK

Several novel and established location and resource aware overlay protocols were compared in this paper using two very specific scenarios. While the multi-tiered hierarchical approach HRM4 provided the best combination of node lifetime and lookup deliverability, its location-awareness is not suitable for MANETs without the further integration of PIS or PRS (proximity-aware identifier/route selection). RBFM demonstrated the most stable behavior with respect to node lifetime, lookup deliverability, and location-awareness, often outperforming two-tiered approaches. The two-tier approaches performed well on many measures, but failed to provide the robustness required for a system with high churn rates. Thus, the improvement of these approaches' robustness could provide promising resource and location aware alternatives. The evaluation's observations build a foundation for future work with more complicated and realistic scenarios, for example with upper bounds on the permissible load per time unit for varying resource levels, or the development of protocol-specific replication geared toward each approach's specific strengths and weaknesses.

7. REFERENCES

[1] F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri. Chr: a distributed hash table for wireless ad hoc networks. In *ICDCS '05*.

[2] M. Artigas, P. Lopez, J. Ahullo, and A. Skarmeta. Cyclone: A novel design schema for hierarchical dhts. In *P2P'05*.

[3] M. S. Artigas, P. G. Lopez, and A. F. Skarmeta. A comparative study of hierarchical dht systems. In *LCN'07*.

[4] I. Baumgart, B. Heep, and S. Krause. Oversim: A scalable and flexible overlay framework for simulation and real network applications. In *P2P'09*, 2009.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM '04*, 2004.

[6] F. Bustamante and Y. Qiao. Friendships that last: Peer lifespan and its role in p2p protocols. In *Web Content Caching and Distribution*. 2004.

[7] C. Cramer and T. Fuhrmann. Proximity neighbor selection for a dht in wireless multi-hop networks. In *P2P '05*.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *SIGCOMM'04*.

[9] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris. Designing a dht for low latency and high throughput. In *NSDI*, 2004.

[10] M. El Dick, E. Pacitti, and B. Kemme. Flower-cdn: a hybrid p2p overlay for efficient query processing in cdn. In *EDBT '09*, 2009.

[11] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB '04*, 2004.

[12] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *ICDCS'04*, 2004.

[13] L. Garces-Erice, E. W. Biersack, P. A. Felber, K. W. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *ICPDCS*, 2003.

[14] J. Garcia-Luna-Aceves and D. Sampath. Scalable integrated routing using prefix labels and distributed hash tables for manets. In *MASS '09*, 2009.

[15] P. B. Godfrey and I. Stoica. Heterogeneity and load balance in distributed hash tables. In *INFOCOM*, 2005.

[16] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *SIGCOMM '03*.

[17] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, 1997.

[18] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for peer-to-peer systems. In *SPAA'04*, 2004.

[19] R. Kummer, P. Kropf, and P. Felber. Distributed lookup in structured peer-to-peer ad-hoc networks. In *DOA*. 2006.

[20] B. Maniymaran, M. Bertier, and A.-M. Kermarrec. Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks. In *ICDCS '07*, 2007.

[21] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, 2002.

[22] G. Millar, E. Panaousis, and C. Politis. ROBUST: Reliable overlay based utilization of services and topology for emergency MANETs. In *Future Network and Mobile Summit '10*.

[23] H. Pucha, S. Das, and Y. Hu. Ekta: an efficient dht substrate for distributed applications in mobile ad hoc networks. In *WMCSA '04*, 2004.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *SIGCOMM'01*, 2001.

[25] 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 *WSNA '02*, 2002.

[26] S. Ren, L. Guo, S. Jiang, and X. Zhang. Sat-match: a self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks. In *IPDPS'04*, 2004.

[27] L. Ribe-Baumann. Combining resource and location awareness in dhts. In *LNCS*, 2011.

[28] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *MMCN*, 2002.

[29] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, 2001.

[30] Z. Tian, X. Wen, Y. Sun, W. Zheng, and Y. Cheng. Improved bamboo algorithm based on hierarchical network model. In *CCCM '09*, 2009.

[31] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. *SIGCOMM*, 2003.

[32] Z. Xu, R. Min, and Y. Hu. Hieras: A dht based hierarchical p2p routing algorithm. In *ICPP'03*, 2003.

[33] A. Yu and S. Vuong. A dht-based hierarchical overlay for peer-to-peer mmogs over manets. In *IWCMC '11*, 2011.

[34] T. Zahn and J. Schiller. Madpastry: a dht substrate for practicably sized manets. In *ASWN '05*, 2005.

[35] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Technical report, UC Berkeley, 2001.

[36] S. Zoels, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical dht design. In *P2P '06*, 2006.