Combining Resource and Location Awareness in DHTs

Liz Ribe-Baumann

Ilmenau University of Technology
liz.ribe-baumann@tu-ilmenau.de

Abstract. Distributed hash tables are designed to provide reliable distributed data management, but present challenges for networks in which nodes have varying characteristics such as battery or computing power. Assuming that nodes are aware of their resource availability and relative network positions, this paper presents a novel distributed hash table protocol which uses nodes' resource levels to remove load from weak nodes, whose overuse may cause delays or failure, while using nodes' positions to reduce cross-network traffic, which may cause unwanted network load and delays. This protocol provides nodes with links that are physically near with high resource availability, and simultaneously provides scalability and an $O(log(N))$ routing complexity with N network nodes. Theoretical analysis and simulated evaluation show significant decreases in the routing and maintenance overhead for weak nodes, the physical distances that lookups traverse, and unwanted node failures, as well as an increase node lifetime.

1 Introduction

Distributed hash tables (DHTs) have received much attention over the past decade as an efficient, reliable approach to store large amounts of data in a distributed fashion. Well established DHTs such as the Content Addressable Network (CAN) [18] and Chord [21] were designed to route efficiently on homogeneous networks without location information. Since DHTs do not inherently differentiate between nodes' characteristics (for example, group associations or robustness), building a DHT on a heterogeneous network comes with many challenges. Moreover, DHTs typically route messages along nodes independent of their locations (causing unnecessary lookup delays and cross-network traffic) and must be specially designed to use location awareness. This paper considers systems on which DHTs benefit from both heterogeneous node treatment and location awareness.

Take for example a cloud environment, in which nodes have varying computing power and network connections: Nodes with high computing power should obtain more load than nodes with low computing power while routing hops should follow low-latency links in order to reduce cross-rack traffic and shorten lookup times. Or consider a second example in which widespread wireless nodes run on battery power (e.g. smart phones which are recharged at regular intervals) and maintain a DHT using an intact infrastructure: Nodes with high power availability can handle more load than nodes with very limited power availability while latencies between nodes vary greatly depending on nodes' up and down links. While in both scenarios all nodes may cooperatively share the storage load, reducing the maintenance and routing load on weaker nodes would improve the networks' performance. Moreover, in the second case, minimizing low power nodes' maintenance and routing load can help to lengthen nodes' lifetimes between recharging. A homogeneous use of the nodes would ultimately result in shorter battery lifetimes and thus higher failure rates of low power nodes, thus effecting both the network's robustness and overall storage capacity. In this paper, we use this example of nodes with varying power availability as our point of reference. However, heterogeneous networks of nodes with varying bandwidth, time-to-live, computing power, or other suitable measures can be treated analogously, and therefore, our considerations are formulated in the general terms of network nodes' "resource availability."

The three main approaches to addressing heterogeneous node capabilities have been the use of hierarchical DHT structures, virtual nodes, or node movements within the identifier space, aiming to balance either communication or storage overhead. However, these approaches are difficult to adapt to our scenario. Firstly, the structural approaches which are capable of reducing the communication overhead of low resource nodes incorporate only two resource levels: "have" or "have not" (for example [1,10,25]). Secondly, the virtual nodes and node movement approaches, which allocate varying quantities of data to each physical node, actually introduce more maintenance overhead and churn into the network. Neither is ideal for a network in which nodes have varying access to resources such as energy or computing power, since they assume that higher resource availability infers larger storage capacity. However, considering that the main communication load in a DHT is accrued from maintenance, it is not nodes' storage capacities that need to be treated heterogeneously but rather the maintenance overhead required of the nodes.

This paper presents a novel resource aware and location aware DHT which reduces low resource nodes' load through routing and network maintenance while providing nodes with links which are physically near. We use two different failure models for testing: one which assumes that a node's failure depends on its resource availability and the number of messages it sends and receives (e.g. battery-power as nodes' resource), and a second which assumes that a node's resource availability is correlated with its failure probability (e.g. node lifetime as nodes' resource). We take a primarily structural, or link-based, approach to this problem, essentially transferring a large portion of maintenance and routing responsibilities to high resource nodes.

Our DHT is based on coordinates similar to and inspired by those used by Vivaldi [5] and structured similar to DHash++ [6], yet, it has significantly different behavior with regard to resources. In contrast to other flat DHTs, we choose links within finger intervals which have suitable combinations of high resource level and low distance (see Figure 1) and adjust the maintenance frequency of each link depending on its resource level. This paper's main contributions are:

ProspectiveLinks[m-1]

NodeID	Phy.Dist.	Res.Level	Res.Dist.
p_2ID	1.6	2	2.6
p_3ID	1.4	1	3.4
p_4ID	0.9	0	3.9
p_1ID	4.1	3	4.1

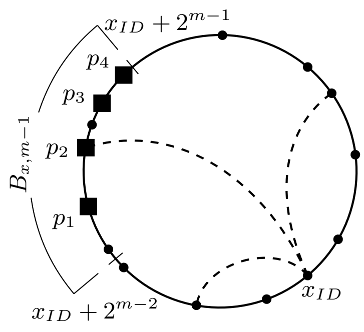

Fig. 1. Key ring with six nodes in x's $m-1^{\text{st}}$ finger interval $B_{x,m-1}$, four of which x knows in its prospective links list (squares). A finger is established to p_2, the known node with the best resource distance (dependent on distance and resource level) to x.

- A novel flat DHT which incorporates both location and resource awareness;
- analytical observations of its routing complexity, links' resource levels and distances, link failure probabilities, and maintenance overhead; and
- simulated comparisons of node lifetime and message failures to resource naive and location aware DHTs with two failure models.

This paper includes an examination of related work in Section 2; discussion of foundational network conditions and assumptions in Section 3; a description of our novel DHT in Section 4; a brief discussion of analytical results in Section 5; and a comparison of the behavior of our DHT with existing DHTs using simulation in Section 6.

2 Related Work

Well established distributed hash tables such as CAN (Content Addressable Network) [18], Chord [21], and Kademlia [17] were designed to route efficiently without location information. Proximity-awareness has begun to permeate areas related to DHTs, including caching and replication protocols and hybrid overlays [7,16], as well as DHTs' original designs. Location awareness is integrated into DHTs using a combination of proximity-aware identifier selection (PIS, such as Mithos [22] and SAT-Match [19]), proximity-aware neighbor selection (PNS, such as DHash++ [6]), and proximity-aware route selection (PRS, such as Tapestry [24]), and has generally been directed at reducing overall traffic or average round trip times [12].

Now recall that the three main approaches to balancing load in heterogeneous DHTs are the use of hierarchical DHT structures, virtual nodes, and node movements within the identifier space. Within hierarchical DHTs, nodes are often grouped 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 [9], Hieras [23], and Cyclone [2] use routing protocols which tend to route lookups as far as possible within one group before forwarding them on to a different group. While resource availability levels could be used to define groups, current approaches pose several problems: lookup routing would have to start in the (sparse) highest layers in order to relieve weaker nodes from routing responsibilities, thus compromising scalability; low level nodes would primarily maintain links within their group, i.e. to low level nodes, increasing their necessary link maintenance; and location awareness could only be provided within single groups, i.e. resource levels. On the other hand, in hierarchical DHTs based on two-tiered peer capabilities (such as [1,10,25]) where nodes assume the roles of super-peer or leaf-peer, lookups are routed directly from leaf nodes to parent nodes. The parent (or super) nodes are fully responsible for performing lookup routing, completely neglecting the varying nuances of nodes' resource availabilities.

In contrast, with virtual nodes, each physical network node balances its load independently by hosting a varying number of virtual overlay nodes, each with its own set of keys and links [11,15]. And similar to virtual nodes, node movements within the identifier space achieve load balance by adjusting the data that each node stores [3,8]. Nodes with low load choose new nodeIDs that are close to nodes with high load, thus taking over some of their load.

Consider now, for example, a Chord implementation run on servers and smart-phones alike: servers have unlimited energy availability and large storage capacities while smart-phones have very limited energy and storage capacities. Since nodes' delays may vary greatly depending on their connectivity, fingers should clearly be chosen across low latency links (i.e. to "near" nodes). While virtual nodes or node movements could help to suitably distribute data, both would fail to relieve weaker nodes with dwindling energy of costly maintenance and routing responsibilities. On the other hand, the integration of a super-peer structure with "have" or "have not" nodes would invariably over-use or under-use the resources of the energy restricted devices. Subsequent node failures would then lead to a drop in the overall network capacity and thus compromise the network's scalability.

3 Network Assumptions

We assume that each node x has sufficiently correct two dimensional virtual (i.e. not necessarily geographical) network coordinates (x_1, x_2), such as used in Vivaldi [5] for determining latencies. The *physical distance* between two nodes x and y is $d_{phy}(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 + y_2)^2}$. Before proceeding, we establish a definition for nodes' resource availability levels and discuss what failure scenarios and underlying DHT properties this work is based on.

3.1 Resource Availability

Recall that we base this work's resource awareness on nodes with varying battery strength, from cell phones with very limited power to servers with an inexhaustible power source. While we model our analysis and simulations after nodes with varying power availability, the proposed protocol need not be restricted to this use case. It requires only 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, i.e. nodes with identical resource levels have comparable available resources. 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.

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 [4,20] (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:

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

3.2 Node Failure

In analysis and simulation, we approach node failure from two different perspectives: On the one hand, we assume that nodes with higher resource levels have lower failure probability (based on nodes with varying time-to-live); on the other hand, we assume that failure is caused by node activities which drain a node's resources until the node fails (based on nodes with varying power availability). For the later analysis of the first case, we need a node's conditional failure probability distribution $P(F_x | x_R = \ell)$, given the event $x_R = \ell$. For our observations, we again choose a Zipf-distribution and assume that this conditional probability is proportional to $P(R = \ell)$, i.e. $P(F_x | x_R = \ell) = \alpha \cdot P(R = \ell)$. Assuming that there are γ simultaneous node failures, we thus have the constraint:

$$\frac{\gamma}{N} = \sum_{j=0}^{l_{max}} P(F_x | x_R = j) P(x_R = j) = \sum_{j=0}^{l_{max}} \alpha P(x_R = j)^2. \tag{2}$$

Thus, we use $P(F_x | x_R = j) = \alpha P(x_R = j)$ for $\alpha = \gamma / \left(N \sum_{j=0}^{l_{max}} P(x_R = j)^2 \right)$.

3.3 DHT Foundation

We chose to build our DHT on Chord [21] mainly because Chord is the basis of the location aware DHash++ and has a rather simplistic structure. Our protocol is, in essence, an extension and could be adapted for many other DHTs. Analogous to Chord, we use consistent hashing [14] to distribute keys to nodes. Each node x chooses a random (or hashed) nodeID x_{ID} from the binary key space

$0 \dots 2^m - 1$, which is viewed as a ring with key values increasing in a clockwise direction. 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 key κ is assigned to the first node whose nodeID is equal to or succeeds κ on the key ring. The asymmetric *key distance* from a node x (or key) to a node y (or key) via their nodeIDs is:

Definition 1 (Key Distance.). *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 Resource and Location Aware DHT

Our novel DHT, which we call *RBFM* for resource based finger management, uses a flat approach to integrate resource awareness into nodes' links, as opposed to the more typically hierarchical approach. We recall from Section 2 that the typical flat approaches of virtual nodes and node relocation actually introduce additional maintenance overhead. We assume that lower resource nodes have an integral role in storing data and propose a system design which distributes data evenly on all nodes while addressing the maintenance and routing responsibilities of heterogeneous nodes. Heterogeneous data distribution and the for DHTs necessary replication are not included in the scope of this paper (see Section 7). Based on Chord, each node in RBFM chooses its links based on other nodes' key distances, physical distances, and resource levels - choosing for each finger interval a link with a balance of low physical distance and high resource level.

Links. Similar to DHash++ [6] and borrowing the terminology from Chord [21], 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 $i \in \{1, 2, \dots, m\}$. The corresponding node that $x.f[i]$ points to is notated $x.f[i].node$. But while DHash++ chooses the node with the smallest physical distance to x in each finger interval, we chose a node based on both its physical distance *and* resource level, or its *resource distance*. This resource distance is inspired by Vivaldi's [5] distance metric for network coordinates which uses an additional height dimension to distance a node from the entire network. Similarly, we use a node's resource level to distance it from the entire network, ensuring that the lower a node's resource level is, the further it will be distanced from every other node. First, we define each node x's resource height x_h via a resource height function $h : \{0, 1, \dots, l_{max}\} \rightarrow \mathbb{R}^+$ for some stretch constant $c > 0$:

$$x_h = h(x_R) := c \cdot (l_{max} - x_R),\ \ell \in \{0, 1, \dots, l_{max}\}. \qquad (3)$$

Definition 2 (Resource Distance). *The resource distance between nodes x and y with coordinates (x_1, x_2), (y_1, y_2), resource levels $x_R, y_R \in \{0, 1, \ldots, l_{max}\}$, and resource heights $x_h = h(x_R)$ and $y_h = h(y_R)$ is:*

$$d_{res}(x, y) = d_{phy}(x, y) + x_h + y_h.$$

In order to gain information about other nodes' resource distances, coordinates and resource levels are piggybacked on network messages. Each node x maintains a *prospective links* list which contains a list of the k best known nodes in terms of resource distance for each finger interval $B_{x,i}$, $i \in \{1, 2, \ldots, m\}$. Thus, at most k nodes in $B_{x,i}$ with the shortest resource distances to x are saved via their nodeIDs and resource distances to x. When receiving a message from sender y, node x uses y's coordinates and resource level to determine $d_{res}(x, y)$ and update its prospective links list accordingly (see Algorithm 1).

Algorithm 1. Updating prospective links list with $\leq k$ entries

```
procedure SUGGESTPROSPECTIVELINK(nodeInfo)
    finger = getFingerInterval(nodeInfo.key)
    dist = getResourceDist(nodeInfo.coordinates, nodeInfo.resourceHeight)
    if prospLinkList.contains(finger, nodeInfo.key) then
        prospLinkList.updateNode(finger, dist, nodeInfo)
    else if dist < prospLinkList.getFarthestLinkDist(finger) or
            prospLinkList.size(finger) < k then
        prospLinkList.addNode(finger, dist, nodeInfo)
        while prospLinkList.size(finger) > k do
            prospLinkList.removeFarthestLink(finger)
        end while
    end if
end procedure
```

Each node x maintains a *finger table* with one finger $x.f[i]$ in each $B_{x,i}$ for $i \in \{1, 2, \ldots, m\}$: if *prospective links* contains at least one entry for $B_{x,i}$, then the entry with the smallest resource distance is contacted with a finger request (see Figure 1); otherwise, the owner (i.e. successor) of key $x_{ID} + 2^{i-1}$ is contacted as in Chord (see Algorithm 2). An entry from the prospective links list is deleted as soon as it is used for a finger request, ensuring that prospective links are up-to-date and alive. The prospective links list entries are also continually updated with fresh node information, so the network automatically adapts to changes in node resource levels or coordinates. Note that if there is a finger interval that contains no node, then multiple fingers will point to the same node, as in Chord. On the other hand, if there is at least one node in a finger interval $B_{x,i}$, then $x.f[i]$ will point to a node in $B_{x,i}$.

As we will show, the larger i is (i.e. the larger the finger interval), the higher we can expect $x.f[i]$'s resource level to be. This means that high resource level nodes tend to have more incoming fingers than low resource level node s.

Algorithm 2. Establishing and maintaining fingers 1 to $m - 1$

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

Routing. Lookup routing is performed greedily, identical to unidirectional routing in Chord [21]: A node x which needs to lookup a key κ in $0 \dots 2^m - 1$ forwards the lookup to the closest predecessor of κ in its routing table (including its *successor list* and its own nodeID x_{ID}). If x is the closest predecessor, then the key is maintained by x's successor, and the routing is completed after one hop.

Since fingers are not deterministically defined in this approach, allowing fingers to be spaced more irregularly, the expected (and worst case) number of hops necessary to locate a key is higher than in Chord. However, this increase can be expressed as a constant factor, leaving us with the same $O(\log N)$ complexity as in Chord. In fact, simulation results show that the difference in routing lengths is in fact negligible. Note that we use the term *with high probability* to express a probability $\geq 1 - 1/N$.

Theorem 1. *Given a network with N nodes, with high probability, a message is routed from any node to the successor node of any key in $O(\log(N))$ hops.*

Proof. Assume that a node x is to forward a message to the node y responsible for key κ, and let p be κ's immediate predecessor node. We consider how many hops are necessary to reach p. Let p be in x's i$^{\text{th}}$ finger interval $B_{x,i} = [x_{ID} + 2^{i-1}, x_{ID} + 2^i)$. Then either:

- $x.f[i]$ is a predecessor to p, and forwarding to this finger reduces $d_{key}(x,p)$ by at least 2^{i-1}, or
- $x.f[i]$ is a successor to p. Since p is a successor to the key $x_{ID} + 2^{i-2}$, $x.f[i-1].node$ is in the interval $[x_{ID} + 2^{i-2}, p_{ID}]$. Forwarding to this finger reduces $d_{key}(x,p)$ by at least 2^{i-2}.

We see that the key distance d_{key}(x,p) $\leq 2^i - 1$ is reduced by a factor of at least $2^i/2^{i-2} = 1/4$ for each forwarding. Thus, we are within one key of p after at most $\log(3/4) \cdot m$ steps:

$$1 = 2^m \, (3/4)^{\log(4/3)} \, .$$

Furthermore, after $\log(3/4) \cdot \log N$ steps we are within $2^m/N$ keys of p, and - considering the consistent hashing used to generate nodeIDs and assuming that nodes are uniformly distributed in the keyspace - with high probability there are no more than $O(\log N)$ nodes this keyspace. So p, and with it y, are reached in at most $O(\log N)$ hops. □

Node Joins and Failures. In order to join the DHT, a node x must have valid network coordinates, choose a nodeID and resource level, and contact one participating node. Once x has established links to its immediate predecessor p and successor s on the key ring, p and s send their *prospective links* lists to x, which x uses to initialize its own list, and corresponding keys are transfered from s to x. The node x continually updates its *prospective links* and periodically performs finger maintenance (see Algorithm 2) to establish and maintain its fingers.

Node failure is handled as in Chord and extended to remove a prospective link once a node notices that it has failed. While replication is necessary to ensure data availability for ungraceful node failures, this is not included in the scope of this paper.

Updating Links. Given a dynamic network with frequent node joins, failures, movements, and changing resource levels, links must be updated on a regular basis to uphold the network's routing characteristics. Links to nodes which have failed or no longer have the minimum resource distance in a finger interval must be reassigned; links to newly joined nodes must be established. Note that a node's outgoing fingers do not change when its resource level changes, but its incoming links most likely will since its new resource level will eventually be updated in other nodes' prospective links lists. Since we correlate a node's resource availability with its robustness, we choose the frequency with which a finger f is updated depending on $f.node$'s resource level: Fingers to high resource level nodes require updates less often, since high resource nodes are less dynamic. This reduces the maintenance load for both lower resource nodes, which initiate link maintenance, and high resource nodes, which respond to the link maintenance.

We perform link maintenance on a finger f after a time interval which depends on a reference interval t_{ref} and the resource level $f.node_R$. One possible *finger maintenance interval* $g : \{0, 1, \ldots, l_{max}\} \rightarrow \mathbb{R}^+$ in dependency of a resource level ℓ is, for example $g(x_R) = t_{ref} \cdot (x_R + 1)^2$.

By this function, a finger with resource level 0 would be updated after the interval $time_{ref}$ and a finger with resource level 3 would be updated after the interval $16time_{ref}$. By adjusting the function $g(\ell)$, the developer has a direct possibility to tune the network's degree of robustness and maintenance overhead. While routing robustness and maintenance overhead usually imply a direct trade off, the observation of nodes' resource availability softens this trade off by decreasing the maintenance overhead of specific, not all, links. However, this is only the case when a node's failure or reliability is reflected by its resource level.

5 Analysis

We provide measures that indicate that nodes' fingers in RBFM are robuster and show how weak nodes' maintenance load in RBFM is significantly smaller when compared to the resource naive, location naive Chord and the resource naive, location aware DHash++. Since we do not consider replication protocols, we are not interested in data loss. We expect RBFM to use lower level nodes less for routing, which balances load according to nodes' available resources; to decrease lookups' physical distance traveled compared to Chord (but not DHash++), which decreases cross-network load; to reduce link failure, which improves routing performance; and reduce maintenance overhead, which especially benefits low resource nodes. Dependencies between neighboring nodes' fingers make a global statistical analysis impractical, so we asses these criteria on a finger basis by comparing the resource level, physical distance, failure probability, and generated maintenance messages of nodes' individual fingers. While the former evaluations provide a mere anticipation for the system's expected behavior and require much interpretation, the evaluation of maintenance messages provides a clearer view of the systems' changed load.

Note that for our analytical observations, we assume that nodeIDs are uniformly distributed. In reality, nodeIDs are often a deterministic SHA-1 hash of node's IP addresses, but unless the SHA-1 function is tampered with (which is considered hard to do) it distributes nodes nearly uniformly in the key space.

5.1 Expected Resource Level and Distance of Fingers

We start with the physical distance traveled and lookups' used resource levels by determining the probability distributions of the physical distance and resource level of a random node x's i^{th} finger. Since $x.f[i]$ is looked for in a finger interval of size 2^{i-1}, the larger i is, the more nodes x has to choose $x.f[i].node$ from. Since x chooses the node to which it has the smallest resource distance, this node tends to be physically closer with higher resource availability as i increases. In fact, our analysis confirms that long key distance fingers tend to be physically close nodes with high resource levels (see Figure 2).

We derive these probability distributions for a node x's i^{th} finger using the probability distribution $P(x_R = \ell) = p_\ell$ for the resource levels from (1) and some distribution of nodes with respect to x in the coordinate space. This distribution is expressed as the probability that a random node will have a given physical distance to x. The following theorem gives us the probability distributions for the physical distance to and resource level of a finger chosen by x from a given number of nodes k.

Theorem 2. *Given is a node x and a set S of k nodes with resource levels in $\{0, 1, \ldots, l_{max}\}$. Let $y \in S$ be a node with the minimum resource distance to x in S, $f_D(t)$ be the probability density function (pdf) over all nodes' physical distances, and $F_D(t)$ its cumulative distribution function. Then the probability that y has resource level $\ell \in \{0, 1, \ldots, l_{max}\}$ is*

$$P(R_{k,min} = \ell) = \underbrace{k}_{a.} \underbrace{p_\ell}_{b.} \underbrace{\int_0^\infty \overbrace{\left(1 - \sum_{j=0}^{3} F_D(t + h(\ell) - h(j))p_j\right)^{k-1}}^{d.} f_D(t)dt}_{c.}, \quad (4)$$

and the pdf for y's physical distance $t \geq 0$ is

$$f_{D_{min}}(t) = k f_D(t) \sum_{\ell=0}^{l_{max}} p_\ell \Big(1 - \sum_{j=0}^{l_{max}} p_j F_d(t - h(j) + h(\ell))\Big)^{k-1}. \tag{5}$$

A proof is omitted due to space constraints, but we provide a rough sketch by explaining the terms in (4):

a. Number of possible nodes that may have minimum resource distance.
b. Probability that $y_R = \ell$.
c. Probability that for all nodes $v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$.
d. Probability that, with fixed $d_{phy}(x,y) = t$ and $y_R = \ell$, for all nodes $v \in S/\{y\} : d_{res}(x,v) \geq d_{res}(x,y)$.

While node x's i+1st finger interval contains 2^i keys, it only contains an expected $k = \frac{N}{2^m} \cdot 2^i$ nodes. Choosing the node with the minimum resource distance to x from these k nodes, (4) gives us the probability that x's ith finger has resource level $\ell \in \{0, 1, \ldots, l_{max}\}$ and (5) gives us the pdf of its physical distance to x. Note that it is not possible to make any statements about the fingers' resource levels or physical distances without some assumption about the nodes' distribution within the coordinate space. For this reason, we consider one specific and simple case, where nodes are uniformly distributed around x on a disk of radius r (this means that we observe only the disk's center node):

$$f_D^u(t) = \begin{cases} 2t/r^2 & 0 \leq t \leq r \\ 0 & \text{else} \end{cases}, \quad F_D^u(t) = \begin{cases} t^2/r^2 & 0 \leq t \leq r \\ 0 & \text{else.} \end{cases} \tag{6}$$

To simplify (4) and (5) for this distance distribution, we use a concrete instance of the height function $h(x_R)$ from (3) with $c := r/l_{max}\tilde{c}$, which determines the resource height as a fraction of the network's physical radius $r/\tilde{c}$ multiplied by $(l_{max} - x_R)/l_{max}$. Then using (4), we obtain a probability distribution which is *independent* of r

$$\begin{aligned} P(R_{k,min} = \ell) &= \frac{2p_\ell}{r^2} \int_0^r \Big(1 - \sum_{j=0}^{l_{max}} F_D^u(t + h(\ell) - h(j)) p_j\Big)^{k-1} t\mathrm{d}t \\ &= 2p_\ell \int_0^1 \Big(1 - \sum_{j=0}^{l_{max}} F_D^{u'}(t + \frac{j-\ell}{c \cdot l_{max}}) p_j\Big)^{k-1} t\mathrm{d}t \end{aligned}$$

with

$$F_D^{u'}(t) = \begin{cases} t^2 & \text{if } 0 \leq t \leq 1 \\ 0 & \text{else.} \end{cases}$$

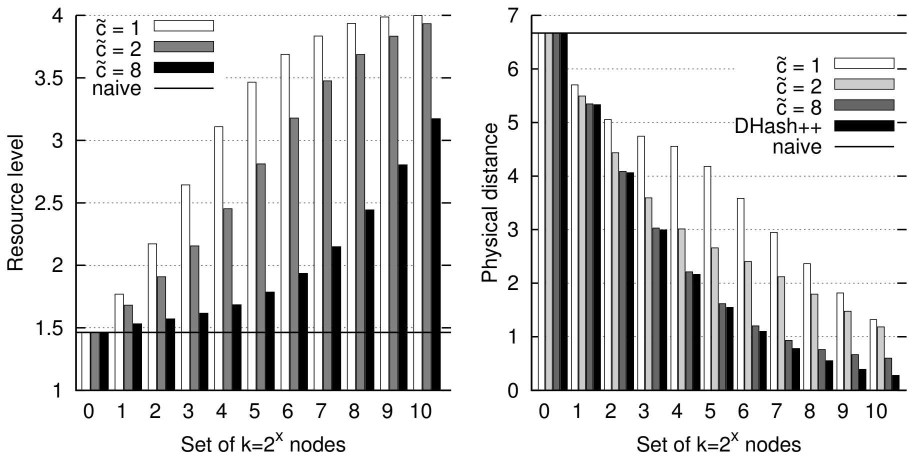

Fig. 2. The expected resource level and physical distance of the node with minimum resource distance from sets of $2^0, 2^1, \ldots, 2^{10}$ nodes are shown for $\mu = 2$ and $r = 10$ along with the respective values for DHash++ and a resource and location naive Chord

The expected values of these probabilities are depicted in Figure 2 for $l_{max} = 3$, specific values of k $(2^0, 2^1, \ldots, 2^{10})$, and $c = 1.5$, as are the corresponding expected values for the physical distance to the node with minimum resource distance. Although we do not expect that x knows all of the nodes in each $B_{x,i}$, the expected number of nodes per finger interval doubles per interval and we presume that each node knows a fair number of nodes per finger interval. A node's $\lceil \log(1/N) + m \rceil^{\text{th}}$ finger interval is the first in which a node is expected to be found, thus, each set of 2^i nodes in Figure 2 corresponds to a node's $i + 1 + \lceil \log(1/N) + m \rceil^{\text{th}}$ finger interval in a network of N nodes.

Figure 2 shows us how stretch affects fingers' resource levels and physical distances, with low stretch ($\tilde{c} = 8$) favoring lower physical distances and high stretch ($\tilde{c} = 1$) favoring higher resource levels in an apparent trade off. For a middle stretch of $\tilde{c} = 2$, a finger's expected resource level are doubled when there are 2^6 random nodes to choose from, while a mere 2^4 nodes are needed to reduce its expected physical distance by more than 1/2. Note that the expected physical distance of a finger in a location unaware Chord is constant and given for $k = 1 = 2^0$, as is the expected resource level of both Chord and DHash++. Interpreting the results for the special case in Figure 2, we expect that resource and location aware fingers cause a higher number of routing hops to be sent across high level, physically close nodes, resulting in less traffic on low level nodes and less cross-network traffic, and ultimately resulting in robuster lookups.

5.2 Failures

Recall that we assume two failure scenarios: nodes which fail according to a given probability distribution and nodes whose resources are depleted with when sending and receiving nodes until node fail. For the first case, we now consider the failure probability of a random node x's fingers given a fixed number of failures where the failure probabilities depend on nodes' resource levels. For the second case, we consider the expected number of maintenance messages for x's fingers per unit of time.

Random failures. Let $f_i = x.f[i].node$ for simplicity and assume that there are $k_i = \lfloor N \cdot 2^{i-m} \rfloor$ nodes in $B_{x,i}$ for $i \geq \lceil \log(1/N) + m \rceil$. Using the conditional failure probabilities for the nodes from (2) and the resource probabilities for the fingers from (4), we have:

$$\begin{aligned} P(f_i \text{ fails}) &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails and } (f_i)_R = j) \\ &= \sum_{j=0}^{l_{max}} P(f_i \text{ fails } |(f_i)_R = j) \cdot P((f_i)_R = j) \\ &= \sum_{j=0}^{l_{max}} P(F_x | x_R = j) \cdot P(R_{k_i,min} = j). \end{aligned}$$

Again using the example distance distribution from Equation 6, Figure 3 shows an example scenario which demonstrates how the stretch affects fingers' failure probabilities. Note that for Chord and DHash++, this probability is constant for all fingers, resulting in fingers which are more likely to fail. Thus, we would

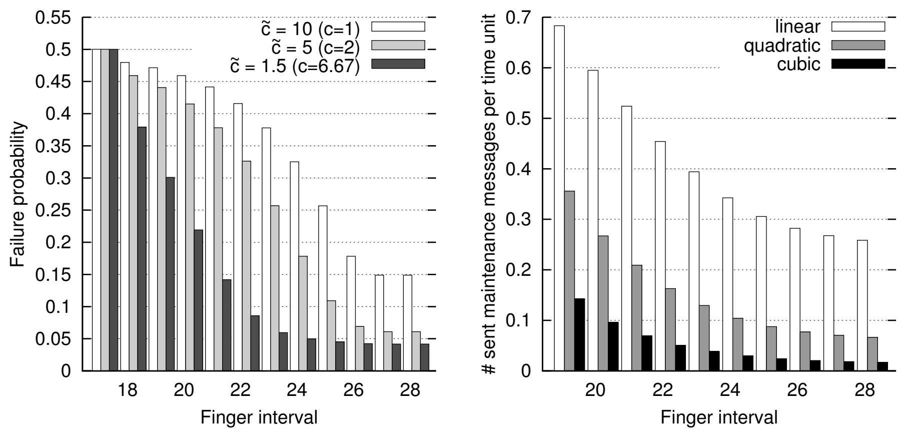

Fig. 3. Scenario with 100000 nodes, 32-bit keyspace, $l_{max} = 3$, $r = 10$, and $\mu = 2$. The probability that a finger in a given interval fails is shown on the left for variable stretch $\tilde{c}$ when half of the network nodes fail. The expected number of sent messages per finger per time unit for linear, quadratic, and cubic finger maintenance intervals is shown on the right, where the 19th finger is the first in which a node is expected.

expect fewer finger failures for the proposed protocol and thus a lower number of lookup failures.

Resource depletion. Restricting ourselves to maintenance messages only, we can use the finger maintenance interval function g and (4) to find the expected number of maintenance messages for a finger f_i per unit of time (see Figure 3):

$$E(\#\text{ messages for } f_i) = E\left(\frac{1}{g(R_{k_i,min})}\right) = \frac{1}{\sum_{\ell=0}^{l_{max}} g(\ell) \cdot P(R_{k_i,min} = \ell)}.$$

By summing up these expected number of maintenance messages over all of a node's fingers, we obtain the expected number of maintenance messages sent by a node during one unit of time. Note that in Figure 3, a constant finger maintenance interval $g(\ell) = t_{ref}$ (i.e. as in Chord) would send one message per unit of time. Similar estimations can also be found for incoming maintenance messages. From Figure 3 we see that for nodes using quadratic finger maintenance intervals, each finger expectedly requires less than 40% of the maintenance messages necessary with a constant finger maintenance interval (and less than 15% for most fingers). Since maintenance overhead accounts for a large portion of the total network load, we expect that this reduced maintenance overhead would lead to a significant increase in the nodes' lifetimes.

6 Evaluation

We evaluated the benefits of RBFM's resource usage by focusing on the direct measures of node lifetimes and lookup failures as well as the indirect measures of the mean resource levels and physical distances used for routing and the total number of maintenance messages sent. The results support our assumption that better finger characteristics do indeed result in improvements in these global behaviors. We used the simulation environment OmNET++ with the overlay framework OverSim [13], which provides implemented overlays such as Chord and Kademlia and uses a coordinate sets with over 200000 coordinates. We extended the functionality of Chord to integrate nodes' resource levels and coordinates, maintain a prospective links list, and use an alternative selection of fingers using the prospective links list. We tested our system in two high-failure scenarios with four resource levels, 10000 nodes with quadratically Zipf-distributed resource availability ($\mu = 2$), nodes' network coordinates provided by the OverSim framework, and a network coordinate diameter of approximately 2000.

In the first scenario, one-third of all nodes fail simultaneously according to a Zipf-distribution on their resource levels, as suggested in Section 3. The second scenario takes a simplistic approach to simulating dwindling battery power: nodes in the bottom three levels are assigned "resources" according to their resource levels - 100 for level 0, 200 for level 1, and 400 for level 2 - which are then decremented for each sent and received message. Top level nodes do not lose resources, due to their inexhaustible resources. A node's resource level is thus decreased with node activity until it is depleted and the node fails. This scenario

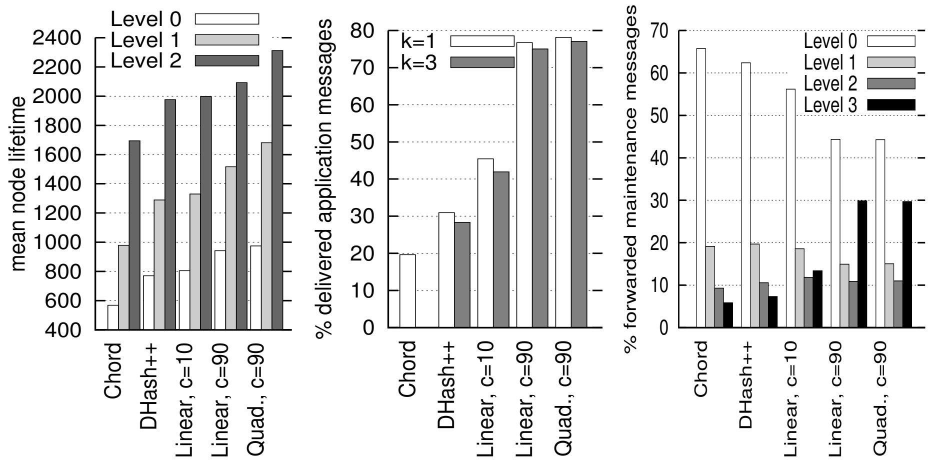

Fig. 4. The leftmost figure shows the mean node lifetimes for varying protocols and resource levels for $k = 1$. The middle figure shows the percentage of delivered application lookups using prospective links lists with one and three entries. The rightmost figure shows the percent of the total *forwarded* maintenance messages that each level forwarded for $k = 1$ and quadratic finger maintenance intervals for RBFM.

is meant to demonstrate the differences between node lifetimes and message loss of the various protocols. All scenarios used a $t_{ref} = 60$ second reference interval between finger maintenance, and RBFM used both linear and quadratic finger maintenance functions, $g(x_R) = t_{ref} \cdot (x_R + 1)$ and $g(x_R) = t_{ref} \cdot (x_R + 1)^2$. Stretch and the number of prospective fingers k were also varied.

A dummy application on each node generated a lookup to a random key every 60 seconds, which served to test the deliverability of messages to random nodes. For both scenarios, the differences between the protocols' average hop counts for these lookups were nearly negligible. The average hop count varied by at the most 5% with 6.5 to 6.8 hops, and the standard deviation of hops per lookup was, at its highest, 1.8.

Due to space limitations, we do not further discuss the results for the scenario in which one-third of all nodes simultaneously fail, but one result is noteworthy. Message failures were comparable at around $20 - 25\%$ for Chord, DHash++, and RBFM with a linear finger maintenance interval for stretch $c = 10, 90$, but surprisingly poor $(45 - 55\%)$ for the quadratic finger maintenance function for $c = 90$. This indicates the importance of adequately frequent finger maintenance for all levels when large scale failures can be expected.

In the second scenario, each node reduced its resources by 0.05 for each sent and received message. New nodes were added once nodes had failed to ensure that there were a total of 10000 nodes with a quadratic Zipf-distribution of their resource levels. Thus, a total of between 34000 and 59000 nodes (depending on the protocol) were introduced to the network over the measurement period of 4000 seconds. All results are based on the means of three runs which yielded

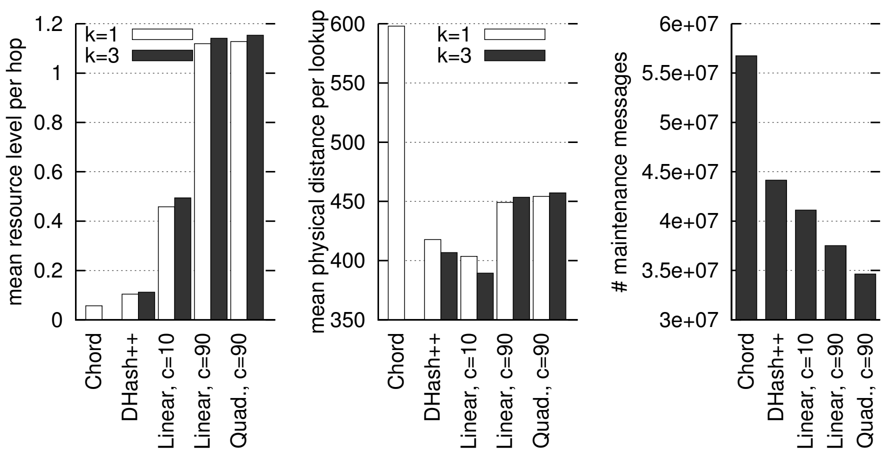

Fig. 5. The left figure demonstrates that the mean resource level per hop with a suitable configuration of RBFM is nearly twenty times that of Chord and ten times that of DHash++. The middle figure shows the mean physical distance traveled by a lookup, the right figure the total number of maintenance messages sent (and forwarded).

negligible variances. The most direct results for node robustness are the mean node lifetimes for various levels and the application lookup deliverability, as shown in Figure 4.

Both results were slightly surprising, in that the resource-naive DHash++ performed significantly better with respect to resources than Chord although we expected both to behave similarly. This improvement can be rationalized by DHash++'s prospective links list. On the one hand, finger maintenance is often routed directly to an existing finger in the prospective links list, generating less maintenance overhead and thus less resource usage (as also seen in Figure 5). On the other hand, nodes with higher resource levels also have longer lifetimes, making then more well known to other nodes by their continually backpacked information. Since this information can propagate longer, these higher level nodes will invariably take up a larger portion of the prospective fingers lists and thus other nodes' fingers, giving it an indirect resource-awareness which improves lookup deliverability.

The remaining results reflect our analysis, with lower lookup failure and higher node lifetimes for RBFM configurations. Note that the comparative values for the lower resource levels are our primary interest, since the highest number of nodes belong to these volatile levels. As expected, the highest mean node lifetime is achieved by the RBFM configuration which sends the most infrequent of the tested maintenance messages: the quadratic finger maintenance interval. And contrary to the first failure scenario, this configuration is clearly adequate for this scenario's constant churn as it has the highest message delivery rate. Figure 4 indicates that the percentage of hops routed over lower levels is significantly reduced for RBFM, depending on the stretch. As expected from the analysis, a substantial portion of forwarding responsibilities have been transfered from lower level to higher level nodes using RBFM. Figure 5 further demonstrates the RBFM reduction in maintenance messages, with a decreasing tendency for increasing stretch (i.e. increasing resource-awareness integrated into the resource distance) and infrequent finger maintenance interval.

The mean resource level per hop and mean physical distance traveled of all application lookups are shown in Figure 5. Note the increase in mean resource level of DHash++ compared to Chord, as mentioned above. While both measures appear to depend on the stretch, as opposed to the finger maintenance interval, RBFM with a linear finger maintenance interval and low stretch actually has a lower mean physical distance traveled than DHash++. This is surprising, since we consider resource awareness and location awareness to be trade offs, and provides motivation for further study. On the other hand, the increase in mean resource levels per hop in Figure 5 surpassed our expectations from the analysis. This increase is due to a combination of the higher resource fingers as in Figure 2 and the fact that higher fingers are used more frequently in routing.

7 Future Work

Using analytical examinations, we have shown that our proposed protocol builds more robust links and significantly reduces the maintenance and routing load on weak nodes. In addition to confirming these findings, simulation has shown that, depending on the scenario, our protocol offers better routing robustness while reducing the overall maintenance overhead and shifting the remaining overhead from weak to strong nodes. This furthermore increases the mean node lifetime in a battery-powered network scenario. RBFM provides an opportunity to allocate storage and query load to low resource nodes while preventing delays and protecting nodes from failure due to maintenance and routing load. The protocol developed here is essentially an extension of Chord and could be considered for other similar DHTs. Future work should consider data replication within this extension and the feasibility of developing a hierarchical DHT to obtain similar resource and location awareness.

References

1. Artigas, M.S., Lopez, P.G., Skarmeta, A.F.: A comparative study of hierarchical dht systems. In: Proceedings of the 32nd IEEE Conference on Local Computer Networks, pp. 325–333 (2007)
2. Artigas, M., Lopez, P., Ahullo, J., Skarmeta, A.: Cyclone: A novel design schema for hierarchical dhts. In: IEEE P2P 2005, pp. 49–56 (2005)
3. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-attribute range queries. In: SIGCOMM 2004, pp. 353–366 (2004)
4. Bustamante, F., Qiao, Y.: Friendships that last: Peer lifespan and its role in p2p protocols. In: Douglis, F., Davison, B. (eds.) Web Content Caching and Distribution, pp. 233–246. Springer, Netherlands (2004)
5. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. In: SIGCOMM 2004, pp. 15–26 (2004)

6. Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a dht for low latency and high throughput. In: Proceedings of the 1st NSDI, pp. 85–98 (2004)
7. El Dick, M., Pacitti, E., Kemme, B.: Flower-cdn: A hybrid p2p overlay for efficient query processing in cdn. In: EDBT 2009, pp. 427–438 (2009)
8. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB 2004, pp. 444–455 (2004)
9. Ganesan, P., Gummadi, K., Garcia-Molina, H.: Canon in g major: Designing dhts with hierarchical structure. In: ICDCS 2004, pp. 263–272 (2004)
10. Garcés-Erice, L., Biersack, E.W., Felber, P., Ross, K.W., Urvoy-Keller, G.: Hierarchical Peer-to-Peer Systems. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 1230–1239. Springer, Heidelberg (2003)
11. Godfrey, P.B., Stoica, I.: Heterogeneity and load balance in distributed hash tables. In: IEEE INFOCOM, pp. 596–606 (2005)
12. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of dht routing geometry on resilience and proximity. In: SIGCOMM 2003, pp. 381–394 (2003)
13. Baumgart, I., Heep, B., Krause, S.: Oversim: A scalable and flexible overlay framework for simulation and real network applications. In: IEEE P2P (2009), `http://www.oversim.org/wiki`
14. Karger, D.R., Lehman, E., Leighton, F.T., Panigrahy, R., Levine, M.S., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: STOC, pp. 654–663 (1997)
15. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: SPAA 2004, pp. 36–43 (2004)
16. Maniymaran, B., Bertier, M., Kermarrec, A.-M.: Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks. In: ICDCS 2007, pp. 33–33 (June 2007)
17. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the Xor Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 53–65. Springer, Heidelberg (2002)
18. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: SIGCOMM 2001 (2001)
19. Ren, S., Guo, L., Jiang, S., Zhang, X.: Sat-match: A self-adaptive topology matching method to achieve low lookup latency in structured p2p overlay networks. In: IPDPS 2004, pp. 83–91 (April 2004)
20. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of the Multimedia Computing and Networking (2002)
21. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM 2001, pp. 149–160 (2001)
22. Waldvogel, M., Rinaldi, R.: Efficient topology-aware overlay network. SIGCOMM Comput. Commun. Rev. 33, 101–106 (2003)
23. Xu, Z., Min, R., Hu, Y.: Hieras: A dht based hierarchical p2p routing algorithm. In: ICPP 2003, pp. 187–194 (2003)
24. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-are location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley (2001)
25. Zoels, S., Despotovic, Z., Kellerer, W.: Cost-based analysis of hierarchical dht design. In: IEEE P2P 2006, pp. 233–239 (2006)