When Multi-Hop Peer-to-Peer Routing Matters

Rodrigo Rodrigues rodrigo@csail.mit.edu
Charles Blake cb@mit.edu

MIT Computer Science and Artificial Intelligence Laboratory

Abstract. Distributed hash tables have been around for a long time [1, 2]. A number of recent projects propose peer-to-peer DHTs, based on multi-hop lookup optimizations. Some of these systems also require data permanence. This paper presents an analysis of when these optimizations are useful. We conclude that the multi-hop optimizations make sense only for truly vast and very dynamic peer networks. We also observe that resource trends indicate this scale is on the rise.

1 Introduction

Distributed hash tables (DHTs) propose a logically centralized, physically distributed, hash table abstraction that can be shared simultaneously by many applications [1, 2].

Recently, DHTs have been proposed as a basic interface for building peer-to-peer systems [3, 4, 5, 6]. These peer-to-peer DHTs share, in their design, the fact that they are divided into a *lookup* layer, which locates nodes responsible for objects in the system, and a *storage* layer that provides the DHT abstraction.

Most work on peer-to-peer DHTs takes the necessity of routing in a flat address space as a given. In particular, peer-to-peer DHTs (with a single exception [7]) employ small lookup state optimizations that lead to multiple routing hops, typically $O(log\ N)$. While often not explicit, we can speculate that the designers of these DHTs fear that the bandwidth or memory requirements of an approach where every node maintains complete system membership information are overwhelming for a "large scale" membership.

Since levels of routing indirection complicate several aspects of the system (e.g., security, server selection, operation latency) we believe it is useful to quantify what "large scale" means. Our thesis is that, in realistic deployment scenarios for peer-to-peer DHTs, maintaining complete membership information is both possible and beneficial.

The main contribution of this paper is to present an analytic model that allows us to determine when indirection is desirable. To answer this question we need to take into account several variables of the system. The most obvious ones are the membership durations and the size of the system, since these affect the size of the membership state and the bandwidth needed to maintain up-to-date membership information. Other variables are also important, though more implicitly, such as the total amount of data stored in the DHT (assuming that the DHT provides a reliable mapping — other assumptions, along with

applications other than DHTs, are discussed in section 6). This will influence how large and dynamic the membership can be — if the membership is too dynamic, the bandwidth required to maintain data redundancy will overwhelm the peers' capacity [8].

Our analysis allows us to conclude that routing only matters for peer networks in excess of a few tens of millions of nodes. For smaller networks, the feasibility of maintaining *data* requires membership to be stable enough that maintaining full membership is cheap. With such stable membership, lookup indirection is an unnecessary complexity that can even hurt overall performance (e.g., by increasing lookup latency or impeding server selection).

The remainder of the paper is organized as follows. Section 2 presents the model for redundancy maintenance in a peer-to-peer DHT. Section 3 presents the model for the bandwidth cost of maintaining full membership information. Section 4 analyzes when routing needs optimizing, and this analysis is generalized to produce a result independent of churn in Section 5. In Section 6 we discuss possible applications of multi-hop routing schemes (other than peer-to-peer DHTs), and we conclude in Section 7.

2 DHT Data Maintenance

In this section we consider the bandwidth necessary for maintaining data in a peer-to-peer DHT. As mentioned, the bandwidth constrains are going to limit the amount of data in the system, the membership dynamics, and the number of nodes in the system, hence it will also influence the need for a routing substrate. We present a simple analytic model for bandwidth usage that attempts to provide broad intuition and still applies in some approximation to currently proposed systems. A more detailed version of this analysis is presented in a previous publication [8].

2.1 Assumptions

We assume a simple redundancy maintenance algorithm: whenever a node leaves or joins the system, the data that node either held or will hold must be downloaded from somewhere. Note that by *join* and *leave* we mean really joining the system for the first time or leaving forever. We do not refer to transient failures, but rather the intentional or accidental loss of the contributed data. Transient failures are masked by the use of appropriate redundancy techniques. We also assume there is a static data placement strategy (i.e., a function from the current membership to the set of replicas of each block).

We make a number of simplifying assumptions. Each one is *conservative* — increased realism would increase the bandwidth required. The fact that we perform an average case analysis also makes it conservative, since it does not consider the worst-case accidents of data distribution and other variations.

We assume identical per-node space and bandwidth contributions. In practice, nodes may store different amounts of data and have different bandwidth

capabilities. Maintaining redundancy may require in certain cases more bandwidth than the average bandwidth. Creating more capable nodes from a set of less capable nodes might take more time. Average space and bandwidth therefore conservatively bound the worst case – the relevant bound for a guarantee.

We assume a constant rate of joining and leaving. Here also, the worst case is the appropriate figure to use for any probabilistic bound. The average rate is therefore conservative. We also assume independence of leave events. Since failures of networks and machines are not truly independent, more redundancy would be required in practice to provide better guarantees.

We assume a constant steady-state number of nodes and total data size. A decreasing population requires more bandwidth while an increasing one cannot be sustained indefinitely. It would also be more realistic to assume data increases with time or changes which would again require more bandwidth.

2.2 Data Maintenance Model

Consider a set of N identical hosts which cooperatively provide guaranteed storage over the network. Nodes are added to the set at rate α and leave at rate λ, but the average system size is constant, i.e. $\alpha = \lambda$. On average, a node stays a member for $T = N/\lambda$.

Our data model is that the system reliably stores a total of D bytes of unique data stored with a redundancy expansion factor k, for a total of $S = kD$ bytes of contributed storage. One may think of k as either the replication factor or the expansion due to coding. The desired value of k depends on both the storage guarantees and redundant encoding scheme [8].

We now consider the data maintenance bandwidth required to maintain this redundancy in the presence of a dynamic membership. Note that the model does not consider the bandwidth consumed by queries.

Each node *joining* the overlay must download all the data which it must later serve, however that subset of data might be mapped to it. The average size of this transfer is S/N. Join events happen every $1/\alpha$ time units. So the aggregate bandwidth to deal with nodes joining the overlay is $\frac{\alpha S}{N}$, or S/T.

When a node *leaves* the overlay, all the data it housed must be copied over to new nodes, otherwise redundancy would be lost. Thus, each leave event also leads to the transfer of S/N bytes of data. Leaves therefore also require an aggregate bandwidth of $\frac{\lambda S}{N}$, or S/T. The total bandwidth usage for all data maintenance is then $\frac{2S}{T}$, or a per node average of:

$$B/N = 2\frac{S/N}{T}, \quad \text{or} \quad BW/node = 2\frac{space/node}{lifetime} \tag{1}$$

Figure 1 plots some example "threshold curves" in the lifetime-membership plane. These assume a cable modem connection (limited by its 200 kbps upstream connection) and that 25% of the upstream bandwidth is used for data redundancy maintenance. They assume the redundancy factor is $k = 8$. This

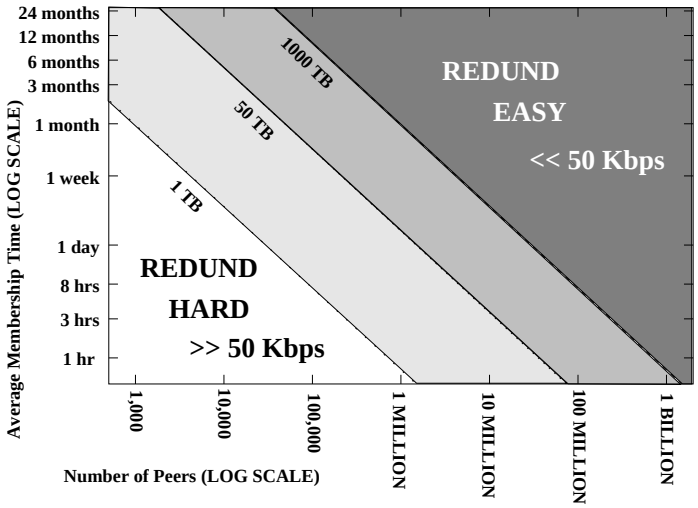

Fig. 1. Log-Log plots for the participation requirements of a cable modem network. Plotted are regions below which various amounts of unique data will incur over 25% link saturation just to maintain the data. These use a redundancy $k = 8$.

is an appropriate encoding rate if availability is over about 75% and we use replication [8].

This figure shows that for a million nodes to cooperatively store a petabyte of data in a DHT, nodes must remain in the system for about one month on average (albeit with possible temporary disconnections [8]).

3 Full Information Lookup

Now we turn to the cost of maintaining complete membership information at each node. The storage cost of maintaining these data structures is negligible. For instance, 10 million IPv4 addresses occupy only 57 MB. What is important are the bandwidth costs which we now analyze.

3.1 Trivial Join Protocol

To bound bandwidth needs, we present a simple protocol for the cost of maintaining full membership information in a peer-to-peer system. The point of this protocol is not to constitute a "complete" system implementation. Instead, we use it for its ostensible simplicity of description, implementation, and analysis. Such analysis will afford us a rough estimate for the main bandwidth costs of membership maintenance, neglecting less costly details such as TCP-layer retransmits.

Our join algorithm is simply this: the joining node downloads the entire member list from anyone and then notifies everyone he is a member. Join notifications are sent continuously until acknowledged from each node. To avoid exorbitant retransmits, the retry time scale is exponentially backed off to some upper limit related to the session time distribution.

This protocol is intentionally simple to ease analysis. A more sophisticated multicast protocol might lower the cost of distributing membership information. The bound we generate is hence quite conservative, since a larger point of this paper is that good system design introduces such fancier protocols and complexity only when required, not proactively for systems of scales which are unlikely to be relevant any time soon.

Assuming that a fraction u of the nodes is unreachable during each notification attempt, this generates the following per-join bandwidth consumption.

- *outbound*:
  $N \times 48 \times (1 + u + u^2 + ...) = \frac{48N}{1-u}$,
  where $48 = 20$ byte node id $+28$ byte UDP header.
- *inbound*:
  acks: $N\times$ 28 bytes. (neglecting IP packet loss)
  member list: $N\times$ 26 bytes,
  where $26 = 20$ byte id $+6$ byte IPv4 address/port.

The average per-node upload or download bandwidth to handle node joins is given by:

$$(outbound + inbound)\frac{\lambda}{N} = (54 + \frac{48}{1-u})N\frac{\lambda}{N} = (54 + \frac{48}{1-u})N/T \qquad (2)$$

3.2 Trivial Leave Protocol

As protocols go, doing nothing at all is fairly simple. All that matters is all remaining nodes to individually notice when a host has been persistently down. This is most naturally done by some staggered or random probing of hosts stretched out over a period of time, τ. Taking partial availability into account, suppose a host must be non-responsive m times – say 4 times to yield a false eviction rate of about 1/256 for unavailability $u = 1/4$. The expected fraction of stale entries is then $P_{stale} = 1 - T/(T + m\tau)$. E.g., $P_{stale} = 1/3$ and $m = 4$ implies that $\tau = T/8$.

A plausible timescale of *true* membership dynamics in data sharing peer-to-peer systems, T, is one-to-several weeks [9]. So, $\tau = 1$ *day* is a plausible target. The bandwidth costs of such a staggered, retrying host eviction is modest. E.g., for $N = 8,640,000$ a refresh time of 1 *day* only costs 100 *probes*/*s* or roughly 20 kbps with 28 byte probe packets.

If N is a bit smaller and queries are randomly distributed then query traffic itself can act as a passive probe at no cost of additional bandwidth. E.g. if the system size is $N = 100,000$ nodes and 64 kbps are used for uploading 8 kB blocks then $\tau \approx 1$ *day*.

3.3 Stale Membership Issues

The principal consequence of the existence of a small fraction of stale membership data is in slightly longer timeouts on *synchronous* "store" operations. When storing data on new nodes the system must timeout store requests that fail, creating a higher operation latency, perhaps ten times the worst case network round-trip – a few seconds of real time. In replica-oriented redundancy, the first few replicas are likely to be stored very quickly yielding some immediate data reliability and availability. Synchronously waiting for the slowest few peers to store provides diminishing returns on the reliability of a data block.

Additionally, reads dominate writes in most applications. It is often a very desirable tradeoff for writes to be a highly concurrent background operation if reads can be made faster. Engineering an entire system around an optimized write latency with no particular application in mind seems off track. E.g., writes to a backup system which occur at off hours to maximize the "user-level" data coherency can take almost as long as one likes while having the particular backup recovery block one wants as soon as possible is more important. Since full-information lookups can surely do at least somewhat better dynamic server selection for "get" operations, the end to end performance of user-relevant operations could be faster.

4 When Indirection Helps

Now we move on to the main question of this paper: When do DHTs profit from multi-hop lookup?

4.1 Combining Our Bounds

To answer this question we combine the results of the two previous sections, first graphically and then algebraically. Figure 2 shows, in the lifetime-membership plane, the intersection between Figure 1 and the region bounded by equation 2. In other words, it shows, on the one hand, when is a peer-to-peer DHT feasible in terms of redundancy maintenance bandwidth (i.e., the region above the dashed lines of Figure 1). On the other hand, it also shows when it is not feasible to maintain complete membership information (i.e., the region below the lines defined by equation 2). For the latter region, we plotted two curves. The solid curve assumes that the average membership maintenance bandwidth for node joins saturated 25% of a cable modem uplink speed (50 Kbps), and fixed $u = 0.25$ in equation 2. We may argue that in some scenarios we would like to start optimizing this cost even earlier, so we plotted another curve (marked "$<<$ 1 Kbps") corresponding to an average membership maintenance bandwidth of 1 Kbps. This ignores the costs of handling node departures, as we have shown that this cost is small.

The intersection of these two lines delineates the interesting region where small lookup state DHTs may be an adequate solution, since it is required to

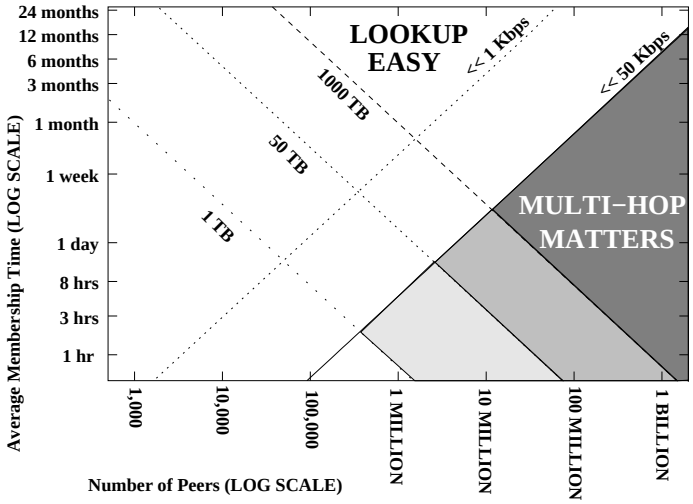

Fig. 2. Putting it all together. The dashed lines from Figure 1 border the feasibility of a peer-to-peer DHT. The lines marked "<< 1 Kbps" and "<< 50 Kbps" border the need for multi-hop lookup. The shaded region corresponds to the deployment scenarios where peer-to-peer DHTs are feasible and multi-hop optimizations may be required.

avoid the large bandwidth consumption of maintaining full membership information (the shaded quadrant marked "multi-hop matters"). The quadrant above the dashed lines and also above the solid line defines when a peer-to-peer DHT is feasible, but multi-hop lookup is not required (marked "lookup easy").

4.2 Data, Pointers, and All That

Figure 2 shows that, for a petabyte of data, DHT feasibility needing multi-hop lookup requires over 10 million participants. Further, for 10 million hosts to store 1 petabyte of unique data, each node will only contribute merely 100 megabytes of unique data, or 800 megabytes of total data (recall we used $k = 8$). This seems like a modest contribution, only 1% of a small disk in a standard PC sold today. More meaningful contributions would require a more stable membership, and thus lessen the need for multi-hop routing.

For a smaller DHT (50 TB), multi-hop DHTs may be interesting if we have to employ a few million nodes to serve the data. To store only *pointers* to a few petabytes of data might require only 1 TB to be stored in the DHT. While storing only pointers does lower the membership scale at which multi-hop optimizations matter, as Figure 2 shows, it only does so for extremely volatile memberships. This then raises the question about how the data itself is being maintained (see Figure 1). If the data is being maintained then a fairly reliable pool of peers is available and could also be used for the lookup service without multi-hop schemes. (The lookup service adds negligible load since looking up a block

uses much less bandwidth than serving a block.) If the data is *not* reliable, then it seems strange to have an indexing service that is much more reliable and available. For the scenario of unreliable data or distributed caching, one of the approaches in Section 6 is more appropriate.

4.3 Hardware Trends

The discussion so far suggests that for systems with the many millions of nodes needed for multi-hop lookup to be a relevant optimization, data maintenance bandwidth constraints prevent using more than a few hundred MB/node. A quick reflection of hardware trends, recapped by Table 1, suggest that multi-hop optimizations will be even less interesting in the future. This table shows how long it would take to upload your hard disk through your network connection for a "typical" user, and how this figure has evolved.

Year	Disk	Home access		Academic access	
		Speed (Kbps)	Days to send	Speed (Mbps)	Time to send
1990	60 MB	9.6	0.6	10	48 sec
1995	1 GB	33.6	3	43	3 min
2000	80 GB	128	60	155	1 hour
2005	0.5 TB	384	120	622	2 hour

Table 1. Disk increased by 8000-fold while bandwidth increased only 50-fold.

Table 1 suggests that disk upload time is getting larger quickly. This implies that if peers are to contribute substantial fractions of their disks then their participation must become more and more stable. This additional stability makes a multi-hop indirection infrastructure even less necessary. This supports the point that multi-hop lookups will not be necessary if one wishes to build a DHT from a large peer population: the more you move "up" on the graph in Figure 2, the less likely it is that multi-hop will matter.

5 Factoring Out Churn

The idea of Section 4 can be generalized to a simple algebraic result *independent of membership turnover*. Equation 1 and Equation 2 can be equated to discover when membership maintenance overhead is about as large as data maintenance overhead:

$$\frac{2S/N}{T} = \frac{(54 + \frac{48}{1-u})N}{T} \approx \frac{118N}{T} \Rightarrow N \approx \frac{S/N}{60 \text{ bytes}}$$

This gives the rough scale at which small-state (i.e., multi-hop) lookup optimizations matter. Per node contributions in the gigabytes implies multi-hop importance scales in the tens of millions. E.g., if $S/N = 3\ GB$, then one only needs multi-hop lookup when $N > 50\ million$.

While it is hard to know for sure, the current Internet is very unlikely to have more than 50 million hosts simultaneously running the distributed service 75% of the time. At the same time it seems very likely that many if not most computers in the current Internet have 3 GB of disk space to contribute. The hardware trends mentioned in Section 4 are therefore quite indicative. As the disk-bandwidth disparity grows, multi-hop lookup systems will become less relevant.

Even if our model is imprecise, estimating only the order of magnitude of membership maintenance, it still seems unavoidable to conclude that current multi-hop DHTs seem relevant, forward-looking designs only when one has either an enormous system scale supporting a small amount of data or draconian membership freshness guarantees.

6 Whither Peer-to-Peer Routing?

The analysis of the paper seems to imply that peer-to-peer routing is only useful in a limited number of situations with very high membership, moderate peer dynamics, and small total data state. However, our analysis is limited to *robust* peer-to-peer DHTs, i.e., DHTs that provide a consistent and reliable mapping from keys to values. Peer-to-peer routing may be extremely useful for other classes of applications. In this section we highlight some relevant examples.

One example are DHTs that do not try to provide a consistent and reliable mapping or do not place replicas randomly. An example of a DHT that falls into this category is Coral [10]. This DHT only stores soft-state — thus it does not require lookups to always retrieve the latest data that was stored under that key — and it uses clustering techniques to create nearby copies of the data. For both reasons, Coral does not incur in the DHT data maintenance costs of Section 2, as the model in question does not apply to it.

The second example is the class of applications, other than DHTs, that take advantage of the peer-to-peer routing topology. An example of such an application is implementing a cooperative multicast infrastructure [11, 12, 13], but other applications exist. These applications do not necessarily store data, and therefore do not incur in the costs of Section 2. Furthermore these applications may find peer-to-peer routing useful for reasons other than saving bandwidth, since they take advantage of the topology that is automatically formed by the routing layer. Therefore the analysis in Section 4 does not apply here, and multi-hop routing can be useful independently of system churn.

Some peer-to-peer storage systems exploit indirection to perform dynamic volatile replica creation along the lookup path. Such caching spreads the load for popular content at the cost of substantial additional bandwidth usage. Effectively, the curves in Figure 1 are shifted away from the origin. While this is a

possibly interesting application of multi-hop lookups, only one level of indirection (i.e., two total routing hops) seems necessary to support this.

7 Conclusion

This paper argues that the feasibility of a peer-to-peer DHT may imply either a relatively stable membership, or an extremely large participation in the system. If the membership is somewhat stable, then the utility of multi-hop lookup optimizations must be questioned.

Our analysis tries to quantitatively bound when multi-hop lookups may be needed. The main conclusion is that this optimization is required only if the system is comprised of more than tens of millions of nodes (under conservative assumptions).

Further, hardware trends indicate that, if peers are to donate significant fractions of the their free storage space to the DHT, then multi-hop lookup optimizations are even less likely to be required in the future.

Acknowledgements

We would like to thank Nick Feamster, Sean Rhea, and Mike Walfish for helpful comments on drafts of this paper.

This research is supported by DARPA under contract F30602-98-1-0237 and by the NSF under Cooperative Agreement ANI-0225660 (`http://project-iris.net`). Rodrigo Rodrigues was supported by a fellowship from the Calouste Gulbenkian Foundation, and was previously supported by a Praxis XXI fellowship.

Bibliography

[1] Gribble, S., Brewer, E., Hellerstein, J., Culler, D.: Scalable, distributed data structures for internet service construction. In: Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000). (2000)

[2] Litwin, W., Neimat, M.A., Schneider, D.A.: LH* - linear hashing for distributed files. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. (1993)

[3] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM. (2001)

[4] Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In: Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). (2001)

[5] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM SIGCOMM. (2001)

[6] Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (2001)

[7] Gupta, A., Liskov, B., Rodrigues, R.: Efficient routing for peer-to-peer overlays. In: Proc. First Symposium on Networked Systems Design and Implementation (NSDI '04). (2004)

[8] Blake, C., Rodrigues, R.: High availability, scalable storage, dynamic peer networks: Pick two. In: Proc. 9th Workshop on Hot Topics in Operating Systems. (2003)

[9] Bhagwan, R., Savage, S., Voelker, G.: Understanding availability. In: Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). (2003)

[10] Freedman, M., Mazières, D.: Sloppy hashing and self-organizing clusters. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03). (2003)

[11] Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A.: SCRIBE: A large-scale and decentralised application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (2002)

[12] Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable networks. In: 3rd International Workshop on Networked Group Communication (NGC'01). (2001)

[13] Zhuang, S., Zhao, B., Joseph, A., Katz, R., Kubiatowicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: Network and Operating System Support for Digital Audio and Video (NOSSDAV'01). (2001)