Peering Peer-to-Peer Providers

Hari Balakrishnan, Scott Shenker, and Michael Walfish
pppp@nms.csail.mit.edu

Abstract

The early peer-to-peer applications eschewed commercial arrangements and instead established a grass-roots model in which the collection of end-users provided their own distributed computational infrastructure. While this cooperative end-user approach works well in many application settings, it does not provide a sufficiently stable platform for certain peer-to-peer applications (*e.g.*, DHTs as a building block for network services). Assuming such a stable platform isn't freely provided by a benefactor (such as NSF), we must ask whether DHTs could be deployed in a competitive commercial environment. The key issue is whether a multiplicity of DHT services can coordinate to provide a single coherent DHT service, much the way ISPs peer to provide a completely connected Internet. In this paper, we describe various approaches for DHT peering and discuss some of the related performance and incentive issues.

1 Introduction

The peer-to-peer revolution introduced the concept of B.Y.O.I. (Bring Your Own Infrastructure), in that the end-hosts receiving service from peer-to-peer applications (*e.g.*, end-hosts sharing files or participating in application-level multicast) were members of an overlay and performed routing and lookup services for other overlay members. The initial distributed hash table (DHT) proposals arose in this context: the purpose of a DHT was to resolve a large, sparse, and flat namespace for *members* of the DHT.

However, the B.Y.O.I. model is not always appropriate for DHTs. For example, researchers have proposed using DHTs, and other flat name resolution mechanisms, to underpin core network services (see [2, 8, 9, 11–13] for a few examples). To be credible, such services cannot depend on the capabilities and caprice of desktop users behind cable modems; rather, these services must run on a set of stable, managed nodes. In addition, as argued in [1, 6], running a DHT is a non-trivial task that requires significant expertise and active oversight. As a result, one school of DHT research, led by Open DHT [1, 6] (the public DHT service formerly known as OpenHash), is proposing a public DHT *service*, *i.e.*, a managed infrastructure supporting a general-purpose DHT. The approach adopted in Open DHT entails two related changes: moving from several application-specific DHTs to a general-purpose DHT service, and moving from the B.Y.O.I. model to a managed infrastructure.

While there might be cases when a benevolent entity (such as NSF) would fund a managed DHT service, it would be preferable if one could arise in a competitive commercial environment. For the Internet, a set of competing commercial ISPs coordinate their activity to provide a uniform Internet "dialtone", and the key issue is how ISPs peer with each other. The question we address here is: can a set of DHT service providers (DSPs) similarly coordinate through peering arrangements to give users a unified, globally coherent DHT "dialtone"?

Our focus here is not on whether such an infrastructure *will* emerge—that will depend on market forces which we cannot divine—but rather on whether such an infrastructure *could* emerge. So, for the purposes of this paper, we assume that market demand for DHT service exists, and we investigate, on a technical and economic level, how DSPs can coordinate to meet this demand. We call the peered collection of DSP providers the P^4 (Peering Peer-to-Peer Providers) infrastructure. In the remainder of this paper, we discuss design possibilities for this P^4 infrastructure as well as the challenges that arise.

These challenges fall into two categories. The technical challenge is to define peering relationships that ensure correct operation of the overall DHT service, allowing customers of one DSP to gain access to data stored by customers of another DSP. The economic challenge is to ensure that DSPs have an incentive to peer (rather than function independently), and to faithfully follow the peering rules. We present a simple design that meets both of these challenges. Thus, we posit that it is possible to offer a coherent DHT service in a commercial and competitive environment. For critical network services, DHTs need not endure the vicissitudes of B.Y.O.I. or government funding but can instead be based on hardened and highly capitalized commercial infrastructures.

2 Design Spectrum

We expect that the P^4 infrastructure supports the following high-level usage scenario, depicted in Figure 1. Customers receive "DHT service" much as they receive DNS service today: the DSP informs its customers of the IP address or name of a host—which we call a *DHT proxy*—and this host handles customers' requests of the P^4 infrastructure. To customers, the DHT proxy is opaque; it

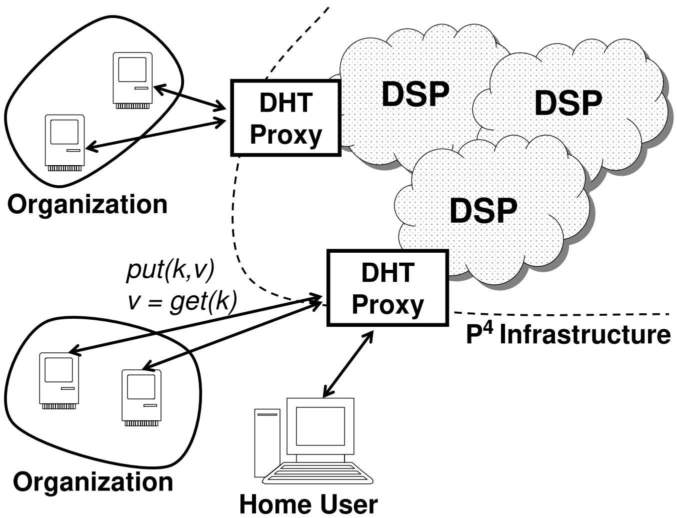

Figure 1: High-level P^4 scenario. Organizations and home users are customers of a DSP; their interface to the P^4 infrastructure is a DHT proxy supplied by the DSP.

might contact another DHT proxy or be one hop away from the P^4 infrastructure. Customer requests are either "puts" of key-value pairs or "gets" of keys. After a customer executes a put request on a key-value pair, (k, v), any other customer of any DSP should receive v in response to a get request for k. In this paper, we do not focus on what happens between customers and their DHT proxies.

We now discuss the goals and design possibilities for a P^4 infrastructure that supports the usage scenario above. Throughout, we are concerned with high-level questions about how DSPs peer with each other rather than with the specifics of network protocols to support this peering.

2.1 Goals

We now list a few of the more crucial design goals; these will help us choose from several design options.

Proper incentives, not perfect security. We do not require that P4 ensures, or even monitors, that DSPs execute their responsibilities properly. Instead, we care only that DSPs have an incentive to do so. This incentive arises if a DSP's malfeasance (such as returning incorrect values) causes harm to its own customers (perhaps in addition to harming customers of other DSPs). If so, then the economic incentives caused by customers switching to or from various DSPs will encourage DSPs to perform their tasks properly. We are not concerned that individual customers may receive bad service from individual DSPs; this situation is analogous to the way today's users of IP are vulnerable to their ISPs.

Flat names. We believe that all keys in the P^4 infrastructure should exist in one flat namespace. In particular, one should not be able to look at a key and deduce which DSP was serving the end-host that put the key. The reason for DSP-independent keys is that if the key *did* identify the DSP responsible for a given (k, v) pair, then the owner of the (k, v) pair would not be able to switch its DSP without invalidating its existing keys.

Flexible tradeoffs between writing and reading speeds. While ideally both writes (puts) and reads (gets) would be fast, in distributed systems one usually sacrifices speed in one to achieve it in the other. Since some applications are read-intensive and others write-intensive, we require that the design allow, on a per-key basis, flexible tradeoffs between these two concerns.

2.2 Designs

We now present four general design approaches and test them against our design goals.[1]

All one DHT. The first design we consider is one in which each DSP contributes hosts to a single, global DHT. The advantage of this scenario is that existing DHT mechanisms work without modification. The disadvantage is that it is a classic "tragedy of the commons". Specifically, a particular DSP reaps all the benefit of bringing in additional customers but only receives a small share of the benefit of providing more resources (nodes) to the DHT. The outcome is likely to be a poorly provisioned infrastructure.

Use administrative separation. To avoid the problem of poor incentives that exists in the previous scenario, we can partition the namespace and have the first few bits of the key, k, identify the DSP "responsible" for k, where "responsible" is defined as "storing the authoritative copy of (k, v)". This model is reminiscent of the Skipnet DHT's [5] use of the top bits of the key to identify the organization in which the key originated. The advantages of this scenario are: (1) everyone knows which DSP is responsible for which key, thereby giving DSPs an incentive to be good P^4 citizens and (2) DSPs would have to store only those (k, v) pairs created by their customers; in response to customer requests for other keys, the DSP could use the information in the key to determine which other DSP to contact. The disadvantage of this approach is that it fails to meet the "flat names" requirement.

The next two designs presume that each DSP maintains its own lookup service and exposes that lookup service to the DSP's customers. Each DSP can implement its own lookup mechanism (presumably, but not necessarily, a DHT), and the internal operations of the various

[1]While we can't prove that these are the only design approaches, they do seem to capture the spectrum of approaches taken for similar problems; see §4.

DSPs can vary widely. In order for DSPs to correctly answer their customers' get queries for all keys in the P^4 infrastructure, DSPs must exchange updates with each other. The difference between the next two designs is whether these updates occur proactively.

Get-broadcasting, local puts. In this design, when a customer executes a put request for a pair (k, v), the customer's DSP stores (k, v) locally. When a customer requests k, the DSP checks if it has stored k. If not, the DSP *broadcasts* the query for k to the other DSPs to ask them about k. As an optimization, the DSP can do this broadcast in parallel with its own lookup. In §3.4, we discuss a further optimization, namely opportunistic caching of (k, v) pairs originating in other DSPs.

Put-broadcasting, local gets. In this design, DSPs *proactively* exchange updates with each other. After a customer puts a (k, v) pair, its DSP updates the other DSPs with information about k. This update can take two forms: the DSP can either tell the other DSPs about the (k, v) pair, or the DSP can tell the other DSPs about k alone, with the understanding that the other DSPs will fetch v on-demand (from the appropriate DSP) when their own customers execute get requests for k.

These last two peering designs address the shortcomings of the first two. As mentioned above, one of our goals is a flexible tradeoff between put and get speeds. Accordingly, we think the last two designs, which comprise three options—get-broadcasting, put-broadcasting of a key, and put-broadcasting of a key-value pair—can coexist. Our assumption is that the user who executes the put request on key k will make the decision about which propagation regime applies to the pair (k, v). This decision is based on the customer's expectations about put and get frequency as well as the cost charged by DSPs.

The three different options involve splitting the resource consumption between puts and gets differently: get-broadcasting has the least bandwidth-intensive put, but the most bandwidth-intensive get; put-broadcasting of a key-value pair is the opposite (most bandwidth-intensive puts, least bandwidth-intensive gets); and put-broadcasting of a key is intermediate. Presumably the charges imposed by DSPs for the various actions, according to whatever pricing scheme they employ, will reflect these differing burdens.

3 Challenges and Questions

Here, we cover the challenges that result from the last two scenarios of the previous section. We emphasize that there are many DHT-related challenges that pertain to our scenario but are addressed elsewhere. The challenges that result from exposing a general-purpose DHT as a service are articulated and addressed by the Open DHT authors [1, 6]. Other challenges, discussed in [7, 13], relate to how, in the absence of cues built into flat names, organizations may offer: fate sharing (the hosts of a disconnected organization should be able to gain access to "local" key-value pairs); administrative scoping (key owners should be able to limit a key-value pair to intramural use); and locality (organizations should have fast access for key-value pairs that are frequently requested by its hosts). These solutions are logically between the DHT proxy and the organization.

3.1 Coherence and Correctness

The P^4 infrastructure must offer to customers a coherent and complete view of the namespace while also letting customers choose their keys. These high-level goals induce two requirements. First, as discussed above, key-value pairs put by customers must be visible to customers of other DSPs. To meet this requirement, DSPs propagate puts and gets (§2.2).

The second requirement is that two customers (of the same DSP or of two different ones) must not be able to own the same key or overwrite each other's key-value pairs. To satisfy this requirement, we borrow Open DHT's [1, 6] three kinds of put requests (to which correspond three types of get requests).

The first kind is *immutable*: k is a secure, collision-resistant hash of v. The second is *authenticated*: putters supply a public key, and getters request not k but rather a (k, a) pair; a is a hash of the putter's public key. For both kinds, the same key (meaning k or a (k, a) pair, depending) should never be claimed by two different owners (unless they are storing the same data, in the first case, or they have access to the same private key, in the second case). These facts are independent of whether the DHT infrastructure comprises one or multiple entities. However, Open DHT's approach assumes that the entire DHT infrastructure is trusted. In contrast, P^4 customers need trust only their own DSPs since the DSPs can check the necessary invariants before accepting updates for *immutable* or *authenticated* key-value pairs.

The third type of put is *unauthenticated*; customers can pick the key and value, but such requests are append-only (to prevent customers from overwriting each other's data). Thus, when a DSP receives a key-value pair from a peer (*e.g.*, on a put-broadcast) for a key it already has, the DSP appends the new value to the existing values associated with the key. Observe that under get-broadcasting, *unauthenticated* puts are only *eventually* coherent;[2] For example, if two customers of two different DSPs put (k, v_1) and (k, v_2), then a local get originating in the first DSP will immediately return (k, v_1), not $(k, \{v_1, v_2\})$.

[2] Under get-broadcasting with TTL-based caching, the other two types of puts are also only eventually coherent, as discussed in §3.4. However, even without caching, the point applies to *unauthenticated* put requests.

3.2 Incentives

As noted earlier, we do not require that the peering arrangements provide perfect security, preventing any malicious behavior on the part of DSPs. We merely require that the incentive to please customers encourages DSPs to behave well. In what follows, the term *data* refers to key-value pairs, *local* puts or gets are those from a DSP's own customers, and *local* data is data stored from a local put. There are four actions that a DSP executes on behalf of customers:

- Respond to local gets (both by answering directly, or requesting the data from other DSPs)
- Respond to external gets (forwarded from other DSPs) for local data
- Process local puts by both storing locally and optionally forwarding to other DSPs
- Process external puts forwarded by other DSPs

In each case, doing the action correctly adds benefit to the local customers, either by providing them with the correct data or by providing others with the local customer's data. If a DSP fails to execute these operations correctly, then—independent of the payment model among DSPs or between DSPs and customers—the customers will become unhappy (if they detect such behavior, which we assume they eventually will if such cheating is widespread).[3]

This discussion of general incentives does not address the question of whether, and how, DSPs would choose to peer. Logically, peering is a pairwise decision in that two DSPs choose to exchange puts and gets. If the two DSPs gain equally, then there will likely be no *settlements* (the common economic term for payments between peers). However, if one of the DSPs benefits substantially more, the DSP benefitting less might demand payment in order to peer.[4] Such settlements would make peering more complicated because they would require detailed monitoring (as explained at the end of this section).

One might think that when a large and small DSP peer, the benefits would be unbalanced. To investigate this hypothesis, consider two DSPs, a and b, who are deciding whether to peer. Assume: (1) that the cost of peering is negligible compared to the other costs of running a DSP[5] and (2) that the profit of a DSP is proportional to the utility its customers derive from its service (the happier the customers are, the more they are willing to pay). Then, the benefit that accrues to a given DSP from peering is proportional to the sum of the benefits that accrue to the DSP's customers from: being able to read data from the other DSP and having their data read by customers of the other DSP.

To calculate these benefits, we use the following definitions:

- b_p: the average benefit a customer derives from having its data read by another customer
- b_g: the average benefit a customer derives from reading a piece of data
- $n_{a\to b}$: number of gets issued by customers of DSP a for data produced by customers of DSP b
- $n_{b\to a}$: number of gets issued by customers of DSP b for data produced by customers of DSP a

The benefit derived by DSP a from peering is proportional to $n_{a\to b}b_g + n_{b\to a}b_p$. Similarly, the benefit derived by DSP b is proportional to $n_{a\to b}b_p + n_{b\to a}b_g$. The difference in benefits is proportional to

$$\Delta = (b_p - b_g)(n_{a\to b} - n_{b\to a}).$$

If the average benefit to a customer from reading data is the same as the average benefit to a customer from having its data read (*i.e.*, if $b_p = b_g$), then both DSPs benefit the same (*i.e.*, $\Delta = 0$), independent of their size. If b_p does not equal b_g, then we must consider the quantity $n_{a\to b} - n_{b\to a}$. We measure the size of DSPs a and b by number of customers and denote these quantities S^a and S^b. Now, assume that the number of gets issued by the customers of a DSP is proportional to the DSP's size, with constant of proportionality λ_g (so the number of gets issued by customers of DSP a is $\lambda_g S^a$). Now assume further that the fraction of data items in the P^4 infrastructure owned by a DSP's customers is also proportional to the DSP's size, with proportionality constant λ_d (so the fraction of total data items owned by b's customers is $\lambda_d S^b$). Now assume finally that all gets are targeted randomly in the namespace, so the number of gets destined for a DSP is proportional to the fraction of data items its customers own. Then, $n_{a\to b} = \lambda_g S^a \lambda_d S^b$, which is symmetric in a and b. Thus, if the preceding assumptions hold, DSPs benefit equally, independent of their size.

Clearly these assumptions won't hold in practice exactly. However, if they are a reasonable approximation, DSPs might choose to peer without settlements. If the assumptions aren't even close, and settlements are thus required, then monitoring is necessary (if DSP a locally serves gets for a key-value pair it received on an update from DSP b, then b has no way to know how many gets were thus served, and a has no incentive to be truthful.)

[3] A DSP can certainly deny a customer access to a strategic key-value pair; the potential for such abuse appears in many customer/provider relationships (including those discussed in §4).

[4] There is a vast economics literature on this two-person *bargaining* problem, where a joint venture benefits two parties unequally. The nature of the solutions doesn't concern us here, except that the literature is unanimous in expecting no payments in the symmetric benefits case.

[5] In practice, this assumption may hold only when the sizes of the two DSPs are the same order of magnitude; a much smaller DSP would incur comparatively more bandwidth cost from peering. However, as discussed in §3.3, we imagine the peering will be done by large players.

The only easily monitored scenario is get-broadcasting with limited caching.

3.3 Scaling

As with ISP peering, put-broadcasting and get-broadcasting do not scale to a large, flat market structure. However, just as in ISP peering, we assume that a forest structure will arise, wherein: a small number of top-level providers peer with each other; it is these top-level providers that do put- and get-broadcasting; and these top-level providers have "children" that are themselves providers (and may offer a different level of customer service). A child has two options. It can either *redirect* customers' put and get requests to a top-level DSP; alternatively, by sending and receiving updates via its parent, it can maintain a local lookup service.

3.4 Latency

We discuss end-to-end latency experienced by customers for put and get requests. For put requests, the DHT proxy supplied by the customer's DSP checks that any required invariants hold (see §3.1 and [1]) and immediately returns an error or success code to the customer. If the key is a put-broadcast key, the DSP will propagate the put request to its peers in the background. Put requests do not suffer from high latency.

For get requests, we separately consider the three propagation regimes: get-broadcast, put-broadcast of the key, and put-broadcast of the key-value pair. For get-broadcast keys, DSPs perform opportunistic, TTL-based caching (with the TTL set by the putter). Thus, the first time a DSP receives a get request for such a key, the lookup may have high latency since the DSP has to contact the other DSPs. Subsequent lookups will be local to the DSP but then this key-value pair may be stale. (To avoid this staleness, the putter can use one of the two put-broadcast types, which presumably require more payment.) For put-broadcast keys, if the key k is broadcast without the value, v, then, as described in §2.2, all of the DSPs will store both k and a pointer to the DSP that actually has v. The latency situation here is similar to the latency in the get-broadcast regime (in both cases, a get causes a DSP to contact, and wait for, at least one other DSP). Finally, if both the key and value are put-broadcast, all of the DSPs will have copies of (k, v), so latency will not suffer.

Application software acting on behalf of putters can implement an adaptive algorithm that, for each key, decides which propagation regime is optimal, given the costs charged and benefits received.

4 Related Work

The observation that for-profit DSPs could peer to form a federated DHT infrastructure exposing a global namespace was briefly mentioned in [2, 13], but no such mechanism was described. This paper fills that void. We now discuss existing federations (arising in different contexts) that present a coherent view of a namespace or of an infrastructure.

Today's competing ISPs federate by exchanging routes with each other to create a global IP dialtone for their customers. The economic incentives in this federation are similar to what we imagine for the P^4 infrastructure, though the technical challenges differ. ISPs can aggregate (while DSPs cannot) the information they exchange with each other, but ISPs must also apply (while DSPs need not) complex policies about what information to expose to peers. Also, no equivalent of get-broadcasting exists with ISPs; route changes are distributed proactively.

The namespace of the Domain Name System (DNS) is hierarchical, and the "providers" of the resolution service are coded directly into the names. These "providers" need not exchange updates, since, on a get request (*i.e.*, a DNS lookup), the end-host knows how to find the responsible provider.

The literature on content internetworking [4, 10] describes scenarios in which content distribution networks (CDNs) peer to exchange cached copies of Web objects. Those scenarios and P^4 face similar technical challenges in terms of how entities relate to each other (*e.g.*, when and how to exchange updates) but, within an entity, the solutions differ. CDNs do widespread caching of Web objects that have DNS names, and the hosts comprising a CDN may offer application-specific functions such as serving media files. In contrast, DSPs are optimized for lookup and insertion of small values that have flat names.

While the above federations rest on commercial relationships, other federations rely on a combination of altruism and shared purpose (*i.e.*, the participants are directly interested in each other's data). These non-commercial federations include cooperative Web caches (see citations in [14], esp. [3]), Usenet, and peer-to-peer file sharing networks.

5 Summary

The peer-to-peer revolution, with its B.Y.O.I approach, was a radical departure from the more conventional ways of funding infrastructure. However, commentators on the subject sometimes failed to separate the technical innovations introduced by these peer-to-peer designs—achieving flat name resolution in an unprecedentedly scalable and reliable way—from their economic novelty. In this paper we asked whether one can harness the technical properties of these peer-to-peer designs, specifically DHTs, in a more conventional economic setting.

Our analysis suggests that one can. As we describe, there are peering arrangements that result in a uniform DHT dialtone (for customers) with proper incentives (for DSPs). However, these peering arrangements are a necessary but not sufficient condition for commercially provided DHT service. The market for such DHT service depends on the success of prototypes such as Open DHT [1, 6], which in turn will depend on the prevalence and popularity of applications based on a DHT service.

Acknowledgments

We thank Sean Rhea for useful comments. This research was supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660, British Telecom, and an NDSEG Graduate Fellowship.

References

[1] http://opendht.org.

[2] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the Internet. In *ACM SIGCOMM*, Aug. 2004.

[3] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *USENIX Technical Conference*, Jan. 1996.

[4] M. Day, B. Cain, G. Tomlinson, and P. Rzewski. A model for content internetworking (CDI), Feb. 2003. RFC 3466.

[5] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, Mar. 2003.

[6] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring adoption of DHTs with OpenHash, a public DHT service. In *3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2004.

[7] A. Mislove and P. Druschel. Providing administrative control and autonomy in peer-to-peer overlays. In *3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2004.

[8] R. Moskowitz and P. Nikander. Host identity protocol architecture, Sep 2003. draft-moskowitz-hip-arch-05, IETF draft (Work in Progress).

[9] V. Ramasubramanian and E. G. Sirer. The design and implementation of a next generation name service for the Internet. In *ACM SIGCOMM*, Aug. 2004.

[10] P. Rzewski, M. Day, and D. Gilletti. Content internetworking (CDI) scenarios, 2003. RFC 3570.

[11] R. van Renesse and L. Zhou. P6P: A peer-to-peer approach to Internet infrastructure. In *3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2004.

[12] M. van Steen and G. Ballintijn. Achieving scalability in hierarchical location services. In *26th International Computer Software and Applications Conference*, Aug. 2002.

[13] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Mar. 2004.

[14] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperate Web proxy caching. In *17th ACM SOSP*, Dec. 1999.