On the Consistency of DHT-Based Routing

Jayanth Kumar Kannan
Matthew Chapman Caesar
Ion Stoica
Scott Shenker

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-22
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-22.html

January 31, 2007

On the Consistency of DHT-Based Routing

Paper ID #378, 14 pages

ABSTRACT

DHT-based routing has been proposed recently in the literature and this new routing paradigm promises several advantages (*e.g.,* decreased state requirement) over conventional routing protocols. However, the robustness of such schemes is questionable given the difficulty of maintaining even overlay DHTs on Planetlab. In this work, we seek to address this issue head-on by taking the first steps towards a theory of the consistency of DHT-based routing. We do so by proposing the first *fully-decentralized maintenance algorithm* for DHT-based routing and then prove that it guarantees the property of *eventual consistency*. We then go on to explore a generalization of DHT-based routing, called a rendezvous service, and explain how routing protocols built using a rendezvous service may offer better convergence properties as compared to DHT-based routing. Although it is presently unclear whether DHT-based routing will ever be deployed in the Internet, we address one of the main technical objections against them: their perceived lack of robustness.

1. INTRODUCTION

DHT-based routing (UIP [1], VRR [2], ROFL [3]) is a new routing paradigm that seeks to leverage DHT algorithms for the purposes of routing. Whereas standard Distributed Hash Tables (DHTs) *require* an underlying routing protocol (such as IP) in order to function, DHT-based routing schemes leverage DHT algorithms in order to *provide* routing.

DHT-based routing appears to have several attractive technical features, including close to $O(1)$ or $O(\log(N))$ state per host as opposed to at least $O(N)$ state per host in conventional routing protocols, minimal overhead under most typical churn conditions [2], acceptable overhead under extreme churn, and low stretch. Despite these advantages, these scheme also invoke a well-justified sense of skepticism in the seasoned networking researcher for the following reason.

The difficulty of maintaining overlay DHTs on Planetlab over the routing service provided by IP is well known [12–14]. Generally, DHTs have come to be considered to be fragile beasts that are not at home in the real-world Internet, for reasons like, non-transitivity and transient connectivity. Thus, expecting DHTs to do the job of routing, the bed-rock of any distributed system that needs to be completely reliable, appears at first glance to be far too optimistic. Their lack of robustness is one of the principal technical objections against DHT-based routing.

In this work, we seek to directly address this concern by taking the first steps towards a theory of the consistency of DHT-based routing. We believe that the consistency aspect of DHT-based routing is ideally suited for theoretical attention because an absolute guarantee on convergence can never be obtained by any empirical means. Our contributions towards such a theory are three-fold.

First, we demonstrate a decentralized DHT-maintenance algorithm that is provably correct in the following sense: after *any series of churn events* involving the failure or reinstatement of any number of links and nodes, any two nodes that have a physical path between them can *eventually* reach each other. The issue of correctness is only addressed in passing in VRR [2] and ROFL [3], which rely on a few centralized nodes to ensure correctness. Our algorithm, on the other hand, is the *first provably correct decentralized DHT-based routing algorithm*. Although our theoretical guarantees leaves much to be desired in terms of its limited semantics (only *eventual* consistency in the presence of *fail stop* failures is guaranteed) and quantitative properties (high convergence time), it certainly serves as strong evidence to the robustness of DHT-based routing.

Second, as it turns out, this exercise in eventual consistency has an unexpected payoff of immediate usefulness. We show that our maintenance algorithm for DHT-based routing can be applied in the context of an overlay DHT (*e.g.,* on Planetlab) to help deal with the non-transitivity issues on the Internet. We first argue that, in order to ensure *strong consistency* in the face of non-transitivity, it is *necessary* for every node to have a list of all nodes in the system. Given this assumption, our decentralized maintenance algorithm can be suitably transferred to this context to guarantee strong consistency. This provides the *first theoretically supported algorithm* to deal with *non-transitivity* issues for overlay DHTs, a problem for which there has been no known theoretical solutions or complete practical schemes so far.

Third, in order to redress the high convergence time of DHT-based routing, which turns out to be a fundamental tradeoff known in the theoretical community [4], we define the notion of a rendezvous service, a generalization of DHT-based routing schemes. This notion helps us in developing new routing protocols with faster convergence properties at the expense of greater state requirement. We will first provide a generic prescription to build a routing service on top of any rendezvous scheme. Then, we will identify existing schemes

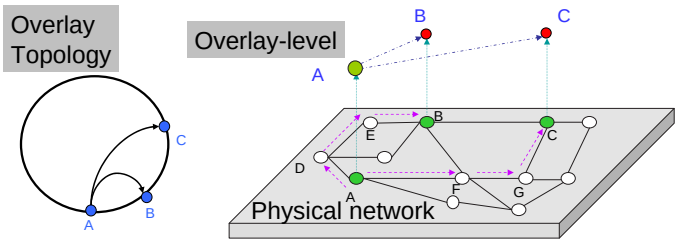

Figure 1: **Model of DHT-Based Routing**

in literature (and propose a new one) as satisfying our notion of a rendezvous service, and build routings protocol based on these rendezvous services. We will then argue that these routing protocols have much improved convergence properties as compared to DHT-based routing. We are the first to observe this general connection between a rendezvous service and a routing service.

We note that while a skeptic may claim that overlay-level DHTs failed to work despite being supported by theoretical proofs of convergence, this misses the point: those theoretical proofs assumed that Internet did not suffer from non-transitive routing. This assumption failed in practice, and hence, those results were simply not applicable. Our theoretical results, on the other hand, take this factor into consideration, and therefore argue for the robustness of DHT-based routing in practice.

Despite these technical arguments for investigating DHT-based routing, a valid question is whether real-life Internet routing protocols would ever incorporate this new paradigm. The current prospects are certainly dismal: in recent years, the networking community has seen several routing proposals (*e.g.,* NIRA [31], HLP [30], FBR [32]) that have had rather limited deployment. Even so, we believe that the paradigm of DHT-based routing bears investigation. Most of these recent proposals were imaginative engineering exercises, that worked with the building blocks of distance vector routing, path vector routing, and link state routing, in order to fashion a scalable Internet routing protocol. DHT-based routing, and its generalization, rendezvous-based routing, on the other hand, we believe, are new building blocks with some superior technical features that should be investigated to support routing designs of the future.

2. MODEL OF DHT-BASED ROUTING

We first discuss a general model (shown in Figure 1) for a DHT-based scheme that captures the three DHT-based routing proposals so far, UIP, VRR, and ROFL.

The Structure: Each node is assigned a random unique identifier from some namespace $[0, 2^n)$ and maintains *pointers* to its *overlay-neighbors*. In this case, A has two overlay-neighbors, B, and C. An *overlay neighbor* for a node is chosen to be a node whose identifier satisfies some relationship with the former's identifier; this exact relationship is prescribed by the DHT algorithm (the region-based design of CAN [6], the exponential splayed out fingers for Chord [7], the tree / hyper-cube structure of Tapestry [9] and Pastry [8]). Unlike the overlay DHT case, where the neighbor is *directly* reachable using an underlying routing protocol, in the DHT-based routing scenario, contacting the neighbor may require going through multiple hops. For this purpose, a *pointer* is maintained for every overlay neighbor as a *source-route* through the network. This source-route consisting of a set of physical links that constitute a path from the node hosting the pointer to its overlay-neighbor. In the figure, for instance, A uses the source-route D, E to reach B and the source-route F, G to reach C. This pointer is periodically refreshed from the originating node, and if it detects that its overlay neighbor is not reachable because of the failure of a on-path link or node or the neighbor itself, it initiates a joining procedure once again.

Routing: When routing to a destination, the node resorts to simple greedy routing: it selects the overlay-neighbor that makes the most progress in the namespace, and then forwards the packet along the pointer to the DHT-neighbor. Forwarding along this pointer can be achieved either through a source-route inserted by the sender (as in ROFL) or through embedded state in the network in the form of incremental source-routes to the overlay neighbor (as in VRR). When the packet reaches the overlay-neighbor, it repeats the same greedy routing process until the packet makes it all the way to the destination. Therefore, routing proceeds at two levels: along the overlay from one overlay neighbor to another, and then from one overlay neighbor to another along the pointer source-route.

Dealing with Churn: When a new node arrives, it discovers and constructs paths to its overlay-neighbors by routing using a physical neighbor that has already joined to act as a proxy. This is possible by the boot-strapping nature of DHTs: a node n can simply route to its own identifier, and discover its overlay neighbors. Since we have already described how to route to a neighbor, when a new node routes to its new identifier (or some function of it), the path taken by this packet as a concatenation of all the pointers it traverses constitutes a source-route to its prospective overlay-neighbor. The new node can use this in setting up a pointer to its neighbor. When a node leaves the network or a link fails, all nodes who had pointers to that node or traversing that link will soon realize the failure of the link, and will effectively rejoin the network. This simple protocol however only works under carefully serialized joins and leaves, which is clearly not realistic.

Path Caches: We discuss VRR's path caches in some detail since it is a useful mechanism we refer to later. The basic idea of a path cache is that nodes on the path of a pointer from A to B, cache (a) the identifier of B (b) the rest of the path up to B (c) optionally, the identifier of A and the reverse path Upton A. This cache is used during routing as follows. When a packet is sent over the pointer from A to B, nodes on the path are free to look at the ultimate destination of the packet C, and then if the have a path cache entry to a node D closer in the identifier space to C than B, they simply "shortcut" the packet to D. These path caches are maintained by soft-state refreshes, and help in significantly reducing stretch in practice. The state required to maintain these path caches should also be included in the state requirements for DHT-based requirement, but this cache is typically so small and its size tunable, so that we omit it in the state calculations.

The list of UIP, VRR, and ROFL, reveals the diverse aspects of DHT-based routing: their allowing (not mandating) all nodes to act as routers, their superior technical properties such as minimal overhead and state, and finally, the fact that they allow the node to be reachable by only its identity in a scalable fashion.

3. EVENTUAL CONSISTENCY

We believe it is important to have theoretical guarantees regarding the consistency DHT-based routing, which at first glance, appears to be fairly complicated for a routing protocol. It is well known that getting overlay-level DHTs running on Planetlab itself has been a ordeal [12, 14], complicated by non-transitivity connectivity issues and the heavily loaded Planetlab machines. Thus, some convincing evidence is required to make the case that DHT-based routing is robust. In work, we prove the correctness of DHT-based routing in the *eventual consistency* sense: if the system is quiescent for some pre-determined time after any series of node/link churn events, then our maintenance algorithms will ensure that the DHT-based routing algorithm will converge correctly.

We note that the correctness issue in the fully-decentralized case has not been dealt with before. UIP, VRR, and ROFL, all deal with correctness in passing and do not formally prove it, and the last two rely on a few reliable nodes for ensuring correctness. Our results, on the other hand, are concerned with the decentralized case, where all node has the same responsibilities, and we offer a theoretical proof of correctness.

The main limitation in our theoretical results on eventual consistency is that the proof works for any sequence of events (nodes and links events, failures and recovery events) provided the system remains quiescent for $O(N^4)$ time afterward (N is the total number of nodes). Though the time requirement appears impractically high and the quiescent requirement rather stringent, we note that this is the time complexity that we have been able to prove by our analytical results; our empirical results indicate the number is much lower in practice. Also, the assumption that the system be quiescent for that duration of time, may appear rather strict; once again, this is more a limitation of our analysis rather than DHT-based routing. Most proofs of practical networking protocols (such as [7]) using the distributed systems formal model of analysis appear to be proofs that make some requirement of this sort. And further, even such proof have been very difficult to obtain for overlay-level DHTs. We now describe our maintenance algorithm and then present its analysis.

3.1 Decentralized Maintenance Algorithm

We now decibel our decentralized maintenance algorithm for DHT-based routing. It borrows from both standard DHT maintenance algorithms (such as Chord's stabilization algorithm) and classical routing protocols (such as distance-vector, path-vector) protocols. Note that applying standard DHT algorithm as such does not work because the fact that the DHT is incorporated into the physical layer brings in other issues (this will become clearer during the description of the algorithm). We will use Chord (with only successors) as the running example during our correctness discussion.

3.1.1 Node State

Before describing the algorithm, we will first detail the state maintained at every node. Each node x (we refer to a node by its identifier x) maintains the following dynamic state: a pointer to the node who it considers to be its successor which includes (a) its successor's identifier, denoted by $S(x)$ (b) a source-route to its successor, denoted by $SSR(x)$. It is clear that if these two pieces of state are set correctly for every node x (*i.e.,* $S(x)$ is the correct successor of x according to the consistent hashing algorithm, and $SSR(x)$ is a valid source-route originating from x and terminating at $S(x)$ that consists of alive nodes and links), then routing is guaranteed to work correctly by the greedy routing properties of Chord. (of course, fingers would also be used for routing, but the correctness of those fingers is ensured by the correctness of the successor pointers [7]). The state at each node is initialized when it comes up as $S(x) = x, SSR(x) = \{x\}$.

3.1.2 Maintenance Algorithm

Algorithm 1 Dynamic State Maintenance At Node x

```
1:  PROCEDURE: update_routing_table()
2:  RETURNS: nothing
3:  // state at node x: Successor S(x), Source Route SSR(x)
4:  // Phase 1: If probe to successor fails, then reset it
5:  if S(x) or SSR(x) is down, reset S(x) = x, SSR(x) = {x}

6:  cand_succs = {(S(x), SSR(x))} // a temporary set used to keep
    track of candidate successors suggested by x's neighbors
7:  // Physical Reconciliation (PR) Phase: Ask phys. ngbrs. for succes-
    sors
8:  for all y such that y is a physical neighbor of x do
9:      // ask y for candidate successor and a route to it
10:     (cand_succ, rt_to_cand_succ) = y.route(x + 1)
11:     // add the candidate and source-route to the set we maintain
12:     cand_succs = cand_succs ∪ {(cand_succ, rt_to_cand_succ)}
13: end for
14: // Strong Stabilization Phase: Ask current successor as well
15: (cand_succ, rt_to_cand_succ) = S(x).route(x + 1)
16: // add the candidate to the set we maintain
17: cand_succs = cand_succs ∪ {(cand_succ, rt_to_cand_succ)}
18: // Phase 4: Invoke Virtual Reconciliation (VR) phase if necessary
19: if (|cand_succs| > 1) then
20:     reconcile_inconsistency(cand_succs)
21: end if
```

We now describe the maintenance algorithm that is required to maintain this state correctly. We will describe it as a periodic action that is initiated by every node, although later, we describe how it can be converted to a mostly on-demand reactive protocol with a very low period background action. We will refer to the neighbors of a node in the physical network as its *physical* neighbors, and the neighbors of a node in the DHT-topology as its *virtual* neighbors.

The algorithm is presented as pseudo-code in Algorithm 1; this procedure is invoked periodically at every node. Of course, each node may invoke this at different times. Its operation is divided into four phases:

- **Detect successor/route failure:** Node x probes its current successor $S(x)$ by sending a probe packet along the current source-route to its successor. If the probe is successful, then no action need be done. Otherwise, it resets its successor and successor source-route to point

to itself.

- **Obtain successors via physical neighbors:** This is called the *physical reconciliation* (PR) phase because, intuitively, it helps a node discover in-consistencies in the opinion of its physical neighbors on its successor, and seeks to resolve them. In this phase, a node proceeds to ask all its physical neighbors y to route to $x+1$. Ideally, all physical neighbors y should return its current successor $S(x)$ and some source-route to it. If, on the other hand, the network is in the convergence process, then its physical neighbors can return different answers. All such candidate successors and routes are retained in a set of candidate successors. Note that the physical neighbor simply invokes the routing algorithm on the identifier $(x+1)$, and the node at which this algorithm terminates is returned as the candidate successor. The physical route (list of all nodes visited on the path, not just overlay neighbors) followed during this process is retained during the packet, and returned along with the identifier of the candidate successor.
- **Obtain successors via current successor:** Node x now asks its current successor $S(x)$ to route to $(x+1)$. It can use its source-route $SSR(x)$ to reach $S(x)$ and then issue this request for routing. This process should again return the successor $S(x)$ upon convergence, but can return a different node until then. Once again, the candidate successor and source-route to it are retained in the set of candidate successors. We refer to this as the *strong stabilization* phase.
- **Reconcile the set of possible successors:** When multiple candidate successors are available in $cand\_succs$, then the node x needs to invoke a reconciliation procedure that anoints one of these candidates as the current successor, and inform the others. This phase is called the *virtual reconciliation* (VR) phase.

The first phase is simple; if the current successor or the route to it has failed, then node x simply elects itself as its successor (since it knows of no other nodes at this point). In this case, the successive phases will ensure that the correct successor is soon chosen. Also, periodic probes are not needed to detect failure if every node along the path maintains semi-soft state and agrees to notify the originating node if the downstream node/link fails. In this case, the last alive node at the end of the last live link on the path will send back a notification if the path goes down. This is the approach used in VRR [2]. The successor pointer is reset on such notification.

The PR phase has the flavor of routing protocols like path-vector where each node periodically obtains a route to every destination from each of its physical neighbors, and then chooses the best route from those. In a very similar fashion, the node x asks its physical neighbors for their "opinion" on what its successor, and adds them to the set of candidate successors. The reconciliation procedure is responsible for picking out the "best" successor. We note that this phase be invoked only at node x only when one of its link goes up or down; periodic execution is not required for correctness as our analysis will show.

The third phase is borrowed from Chord's strong stabilization procedure. This is responsible for ensuring that the "ring" (the graph induced by the successor relationships) does not converge to a so-called loopy cycle (where the node identifiers wrap around more than once) instead of the ideal Chord ring (where the node identifiers wrap around exactly once, and they are well-ordered according to the consistent hashing relationship).

The combination of the PR phase and strong stabilization reflects how our maintenance algorithm borrows from both classical routing protocols as well as DHT maintenance algorithms. The final phase, the VR phase, is achieved by a procedure of linearization, which we now describe.

Algorithm 2 Reconciliation Protocol At Node x

1: PROCEDURE: **reconcile_inconsistency**$(cand\_succs)$
2: RETURNS: nothing
3: Send "Succ Setup" messages to all nodes in $(cand\_succs)$
4: **// confirm any waiting successors or predecessors**
5: For any received "Succ Setup" message from node p_i, send "Succ Confirm" message to p_i, and add p_i to $cand\_preds$
6: For any received "Succ Confirm" message from candidate successor s_i, add s_i to $cand\_succs$
7: **// invoke linearization**
8: **if** $|cand\_succs| > 1$ or $|cand\_preds| > 1$ **then**
9: Let $cand\_succs = \{s_1, \cdots, s_b\}$ and $cand\_preds = \{p_1, \cdots, p_a\}$
10: Linearize $p_1, p_2, \cdots, p_a, x, s_1, s_2, \cdots, s_b$
11: For $i \in \{1, \cdots, (a-1)\}$, Send "Succ Setup" message to p_{i+1} on behalf of p_i
12: For $j \in \{1, \cdots, (b-1)\}$, Send "Succ Setup" message to s_{j+1} on behalf of s_j
13: Send "Succ Setup" to s_1 and "Succ Confirm" to p_{a-1}
14: Set $cand\_succs = \{s_1\}$, $S(x) = s_1$, and $cand\_preds = \{p_a\}$
15: **end if**
16: Go to Step 4

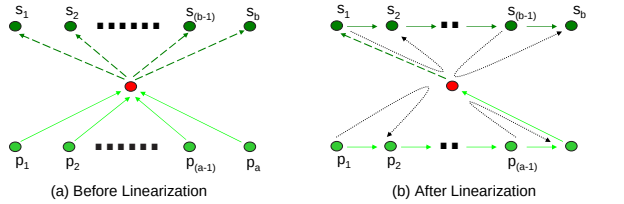

Figure 2: **The linearization procedure: light green (solid) pointers represent pointers from candidate predecessors, while dark green (dashed) pointers represent pointers to candidate successors. The source-routes used in constructing the new pointers are shown by the curved dotted arrows.**

3.2 Virtual Reconciliation: Linearization

The VR phase is shown in Algorithm 2. First, node x sends a "Succ Setup" messages to *all* candidate successors s_i. This successor setup message includes the source route from x to s_i. This message expresses x's intent that it has chosen s_i as its successor. Second, it responds to any incoming "Succ Setup" messages from any nodes p_i with "Succ Confirm" messages, and adds p_i to its list of candidate predecessors. A "Succ Confirm" message implies that x is acknowledging that p_i can choose x as its successor. Note that the source route from p_i to x sent in p_i's "Succ Setup" message. Any "Succ Confirm" messages from any nodes are noted, and such nodes (and their source routes) are added to its list of candidate successors. Note that Step 6 is required because it is possible that a node sends a "Succ Confirm" message to node x with-

out having received a "Succ Setup" message from x; this will become clear later.

It now remains for the node x to choose a single successor and allow a single predecessor. This is achieved by the linearization procedure (shown in Figure 2) locally reconciles the list of candidate successors and predecessors. Each node x first locally orders the set of successors and set of predecessors. It locally computes the sorted ordering of all these nodes $p_1 \rightarrow p_2 \cdots \rightarrow p_a \rightarrow x \rightarrow s_1 \rightarrow s_2 \cdots s_b$ (note that this may involve wrap-around). It then sends a *Succ Setup* message on behalf of p_2 to p_1. This Successor Setup message includes a source route from p_1 to p_2 which is obtained by concatenating the source-route from p_1 to x with the *reverse* of that from p_1 to x. This *Succ Setup* message notifies p_2 that p_1 has chosen it as its successor. This *Succ Setup* message is confirmed by a *Succ Confirm* message from p_1 to p_2. Thus, at the end of this message exchange, p_1 is added to p_2's list of candidate predecessors, and p_2 is added to p_1's list of candidate successors. This process is repeated for all the new successor pointers that are implied by the linearization (note that since p_a and s_1 already point to and from n, so they need not be notified). Each node locally repeats this process until it has in-degree and out-degree of exactly one.

The linearization is similar in spirit to Chord's weak stabilization where a node, on finding a better successor, notifies its current successor of its new choice. The main difference is that linearization handles multiple set of prospective predecessors and successors at one go, and that the source-routes have to be suitably transformed before reconciliation.

The intuition behind linearization is that it performs local reconciliation of all successors and predecessors, while ensuring that all nodes have *at least one* successor and predecessor. This is why, for instance, when node x chooses s_1 as its successor among the candidates s_1, s_2, s_2 is notified that s_1 is a possible predecessor, instead of node x simply dropping s_2. Thus, no nodes will be "lost" from the ring once they are part of it. Further, the linearization procedures ensures that each node keeps finding a better and better successor. Also note that, although the pseudo-code for simplicity requires the node to wait for "Succ Setup" or "Succ Confirm" messages in Step 5/6, in an implementation, this is not necessary. The node can simply go on to the linearization procedure, and process the "Succ Setup" and "Succ Confirm" messages after it is done with linearization. In the limit, it can even process one of these messages every time during linearization (of course, if there are no new messages, linearization will not be invoked at all).

We also wish to note that our maintenance algorithm is fully asynchronous. Each node indulges in the PR phase, strong stabilization, and VR phase periodically without regard to the action of other nodes. Of course, if the PR phase and strong stabilization do not throw any new candidate successors and the node doest not receive any "Succ Setup" messages, then the VR phase is never invoked for that iteration.

3.3 Analysis

In this section, we will define a correctness criterion for DHT-based routing, and then prove that, irrespective of any patterns of churn events, given enough time, the system will eventually converge to its correct state, thus supporting the robustness of DHT-based routing. The correctness criterion basically ensures that if two nodes have even one active path between them, then they can reach other via the DHT. This result applies only under fail-stop failures (*not* byzantine failures) and is of the *self-stabilizing* kind: the system is guaranteed to recover to the correct state even if the current state is arbitrarily corrupted. More precisely:

THEOREM 1. *For a system of N nodes, after any sequence of link and node joins and failures, irrespective of what state the system is currently in, the routing is guaranteed to satisfy our correctness criterion in a maximum of $O(N^4)$ time units and $O(N^6)$ messages. The assumption is that there are no additional churn events during this time.*

The proof technique is to prove that the set of nodes that can talk to one another through the DHT-level pointers converges to the set of nodes that can talk to another via the physical network. In other words, we prove that the "virtual graph" (the set of nodes that can talk successfully to one another using greedy routing on pointers) converges to the "physical graph". The proof technique is presented in a textbook-style *invariant assertion* argument [11]; we prove that a certain invariant holds at every point in time, and then rely on induction, to prove our required result.

We will present our correctness criterion and failure model in detail before delving into the analysis.

3.3.1 Correctness Criterion

We define the following correctness criterion for DHT-based routing: If there is a path between nodes X and Y consisting of alive nodes and active links, X should be able to reach Y by routing through the DHT. In other words, the state should be such that simple greedy routing starting at X is guaranteed to deliver the packet to Y. There are at least two pathological cases where the structure of the DHT may rendered un-usable for routing and further never recover from them. First, certain join orderings may cause a ring to miswrap and contain multiple zeros (this is referred to as a loopy cycle [7]). Second, network-level partitions can cause the ring to partition into multiple pieces, even though the underlying network remains connected.

3.3.2 Failure Model

We assume that nodes undergo fail-stop failures (as opposed to arbitrary Byzantine failures) and come back online at any time. Links are also allowed to fail and recover. In terms of the timing model, we have a choice between the fully asynchronous model (where every node operates at its own speed) and fully synchronous model (where there is a notion of rounds, and all nodes operate in lock-step with each other). In our analysis, we will assume a locally enforced synchronous timing model (similar to that adopted by Subramaniam *et al.* [15]).

In this model, the assumption is that the notion of time is locally enforced by the amount of time that a packet takes to traverse a physical link. Thus, each node counts a time step as the amount of time that a packet takes to traverses a physical link. We assume that the links do not have infinite delay, and thus, a upper bound on the delivery times of packets across

Table 1: The Message Generation Function σ

(Current Phase, State)	Msg Type	(New Phase, State)
(ProbeGeneration, $S(n)$)	ProbeSuccessor	(PhysicalReconciliation,$S'(n)$
(PhysicalReconciliation,$S(n)$)	Routing	(WaitForRoutingReply,$S(n)$)
(StrongStabilization,$S(n)$)	Routing	(WaitForRoutingReply1,$S(n)$)
(Reconciliation,$S(n)$)	SuccSetup	(ProbeGeneration, $S(n)$)

the links in the network is locally used by each node as a time tick. Thus, a packet can be sent across a link in at most one time unit. In our analysis, we ignore packet losses; if the link is active, we assume that the packet can re-transmitted, if required, and sent within one time unit.

3.3.3 Simplifying Assumptions

First, we assume that only successor pointers are maintained by each node. The correctness of routing is dependent only on the correctness of the state of successor pointers, since other state such as fingers and path caches are generated by the successor pointers (which themselves are not influenced by this additional state). Second, we will assume that in the event of failure of a node (or link), its neighbors (or neighbor on the other side of the link) will be able to detect this fail-stop failure within a single time step (as in our model of locally synchronized model). This detection can be accomplished by a periodic ping process between neighbors. For this reason, we assume that when a node sets up a pointer with a source-route to a DHT-level neighbor, the node is notified of failures of this source-route immediately. In other words, we assume that the state of the source-routes is known to the originating node.

3.3.4 Definitions

Following the recipe of the distributed systems analysis methodology (Lynch *et al.* [11]), we explicitly identify the notion of states, messages, message generation functions, and message transition functions within our system, before embarking on the consistency proof. In the rest of the proof, we will use n or x to refer to a particular node.

State Space: The dynamic *state* associated with a node n is denoted by $S(n)$. $S(n)$ consists of two ordered lists $Succ(n) = \{s_1, s_2, \cdots, s_b\}$, $Pred(n) = \{p_1, p_2, \cdots, p_a\}$. $Succ(n)$ includes all the successors of n and $Pred(n)$ contains all the predecessors of n. Note that this state includes all the temporary candidate successors and candidate predecessors maintained in the intermediate steps in Algorithm 1, 2. The state S associated with the set of nodes $\mathbb{N}$ is simply the union of the states associated with all nodes in the system. In other words, $S(\mathbb{N}) = \{S(n) : n \in \mathbb{N}\}$.

Physical Graph: The physical graph, in contrast to the abstract virtual graph that models the dynamic state maintained in the system, captures the state of all the physical nodes and physical links in the system.

Virtual Graph: We use the notion of a *virtual graph* to visualize the state space and state transitions of this system. The virtual graph is the graph induced by the set of successor relationships and predecessor relationships associated with each node. This graph contains a directed edge from node n_1 to node n_2 if $n_2 \in Succ(n_1)$. Note that, this automatically implies that $n_1 \in Pred(n_2)$ because, in Algorithm 2, since a node remembers all the nodes it sent a "Succ Confirm" message to.

In general, each node in a virtual graph can have multiple incoming and outgoing successor pointers since it can maintain several candidate successors and predecessors. Based on our correctness criterion, we define the notion of a *correct* virtual graph. An *correct* virtual graph is one where each node has a single outgoing pointer to its correct global successor on the ring and has a single incoming pointer from its correct global predecessor. By global successor / predecessor, we mean the node considered as the closest successor / predecessor among the set of all *alive* nodes, according to the consistent hashing relationship. The goal of the maintenance algorithm is to ensure that the virtual graph of the system converges to the correct one; in this state, greedy routing based on these pointers (and fingers constructed on the basis of these pointers) will work correctly.

In addition to the notion of correctness, we also say that a virtual graph is *weakly connected* if the underlying *undirected* graph of the *directed* virtual graph is connected. In other words, there is a path from any vertex to another in the virtual graph if the direction of the edges is ignored. Another notion useful in the analysis is the characterization of a *successor-consistent* cycle. A cycle induced by a set of nodes in the virtual graph is said to be *successor-consistent* if the edges among this set match exactly those in the correct virtual graph defined for that particular set of nodes. For example, in a network of 3 nodes, with identifiers from 1 to 6, there can be two successor-consistent cycles at the same time: $(2, 5, 6)$, $(1, 3, 4)$.

Messages: The messages in our protocol are: the "Routing" message (which is used in the PR and strong stabilization phases), the "Routing Reply" message which is sent in response to the "Routing" message, the "Probe Successor" message which is a simple probe, the "Succ Setup" message which are used to request for a successor setup, and the "Succ Confirm" message which confirms the establishment of a node as a predecessor. The "Routing" message does not affect the state of a node; it is simply processed according to the greedy routing algorithm, and the source-route to the concerned node is returned to the requesting node in the form of a "Routing Reply" message. The "Succ Setup" and "Succ Confirm" messages are processed according to the reconciliation procedure, and update the state $S(n)$ accordingly.

Message Generation Function: In order to specify the message generation function σ, apart from the state $S(n)$ of a node, we will also allow a variable $P(n)$, to represent the phase in which the node is in. Initially, the $P(n)$ is set to *ProbeSuccessor*. The message generation function σ maps the current phase and the current state of the node to (a) the messages it should send out (b) the new phase and the new state of the node. A node is initially in the ProbeGeneration phase, where it sends out a probe to its current successor, and updates its state according to the result of the probe. Then, it moves to the PhysicalReconciliation where it sends a Routing request via its physical neighbors, and then, upon receiving Routing Reply messages from them, updates its successor list $Succ(n)$ accordingly, and moves to the StrongStabilization phase. In this phase, it sends out a Routing message

via its current successor, and after receiving a Routing Reply message in response, adds the source-route and candidate successor list to its successor list $S(n)$. It finally moves then to the Reconciliation phase, where it sends all the SuccSetup messages as required by the linearization phase, and finally reverts back to the ProbeGeneration phase.

Message Transition Function: The message transition function π specifies how the state of the node changes in response to incoming messages. It is implicitly involved in the message generation function. We will simply specify it informally since the full function is rather extensive. On receiving a Routing Reply message, at either the end of the PhysicalReconciliation phase or the StrongStabilization phase, the new candidate successor is added to the successor list $Succ(n)$. On receiving a Succ Setup message, the candidate predecessor (and source-route) is added to the predecessor list $Pred(n)$, and a Succ Confirm message is sent to the neighbor. On receiving a Succ Confirm message, the newly confirmed successor is added to the successor list $Succ(n)$.

3.3.5 Analysis

We finally delve into the analysis. The particular scenario we consider is as follows. Consider the case where there has been severe node churn and link flapping for a sustained period of time, at the end of which the network is left in a completely erroneous state. We will show that irrespective of what this state might be, our maintenance algorithm will guarantee that the system eventually converges to the correct virtual graph.

First, for simplicity, we will analyze the algorithm *without* the strong stabilization phase. We will prove that the virtual graph will converge to the ideal virtual graph or a loopy cycle over all the alive and connected nodes in the system. Then, intuitively, we can let the strong stabilization phase operate from this point on. By the properties of the strong stabilization phase [7], then we are guaranteed this loopy cycle converges to the ideal virtual graph. It remains to show that if the strong stabilization phase operates alongside the initial convergence phase, the same convergence is obtained. We will justify this argument later.

Proof Outline: In what follows, we will use N to refer to the number of nodes in the system, and use n or x to refer to a particular node. Under our assumption of local synchronicity, consider the system as it passes through multiple iterations of the PhysicalReconciliation and Reconciliation phases (note that we are skipping StrongStabilization for now). We will model system transitions by following the changes in properties of the virtual graph VG representing the state of the system $S(\mathbb{N})$.

We will associate a *potential function* for every virtual graph VG that roughly corresponds to the "distance" of the graph VG from the correct virtual graph. The proof relies on showing that this potential function has to decrease monotonically. We show that the potential function is upper-bounded by U; this implies that the process has to necessarily stabilize within period U since the value of the potential function can be at most U in the beginning, and it keeps decreasing at every iteration. Thus, this is a classic invariant assertion proof (Lynch *et al.* [11]), since we establish an invariant about the state transition (since the virtual graph is simply an encoding of the state of the system $S(\mathbb{N})$), and thus prove that the state converges to the ideal state.

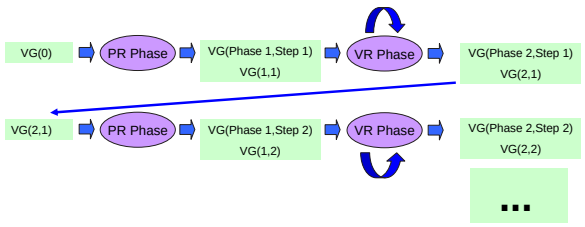

Figure 3: **The State Transitions during the Maintenance Protocol**

Definition of Potential Function: We define the *node potential* for a node n in a virtual graph VG as $\sum_{(n,n')\in VG} d(n,n') + \sum_{(n',n)\in VG} d(n',n)$ where $d(n,n')$ is defined in the distance in the correct virtual graph from n to n' (adjusted by subtracting 1). The offset by one ensures that the node potential for a node that has found its successor is zero. We define the potential function $d(VG)$ for a virtual graph VG as the sum of node potentials $d(n)$ over all nodes n in VG. Since $d(n,n') < N$, this implies that $d(n) \leq N^2$, which in turn implies that, $d(VG) \leq N^3$.

Proof: Figure 3 introduces the notation used to describe the state transitions during the maintenance protocol. Denote by $VG(0)$ the virtual graph (the state) of the system at the end of the stress phase where there is active churn. Since there is no additional churn, we do not need to analyze the ProbeSuccessor phase, since any source-routes discovered during the maintenance will continue to work. Further, as we commented before, we will incorporate the effect of the StrongStabilization phase later. Thus, we can restrict our attention to the Physical Reconciliation and the Virtual Reconciliation phases. The self-arcs on the VR phase reflect that the linearization procedure may be invoked multiple times in Algorithm 2.

Denote by $VG(1,i)$ to be the state at the end of the i^{th} iteration of the Physical Reconciliation (PR) phase, and $VG(2,i)$ to be the state at the end of the i^{th} iteration of the Virtual Reconciliation (VR) phase. This is indicated in the figure for two time steps. Our main theorem follows from the following lemmas:

- **PR Connectivity:** If the physical graph G is connected, then the virtual graph $VG(1,1)$, produced at the end of the PR phase in the first iteration, is weakly connected.
- **VR Phase Retains Connectivity:** If the virtual graph $VG(1,i)$ is weakly connected, then the virtual graph $VG(2,i)$ is weakly connected..
- **PR Phase Retains Connectivity:** If the virtual graph $VG(2,i)$ is weakly connected, then the virtual graph $VG(1,i+1)$ is weakly connected.
- **VR Phase decreases potential function:** Provided $VG(1,i)$ has any node with indegree or outdegree exceeding 1, then $d(VG(2,i)) < d(VG(1,i))$.
- **PR + VR Phase decrease potential function:** $d(VG(2,i+1)) \leq d(VG(2,i))$.

- **Convergence:** If $VG(2,i)$ does not have any node with indegree or outdegree exceeding 1, then it must be a union of successor-consistent or loopy cycles.

The **PR Connectivity** lemma along with the two following lemmas on connectivity implies that the virtual graph $VG(1|2,i)$ is always weakly connected: the property of weak connectivity of the virtual graph is ensured by the PR phase, and is retained by the VR phase. The potential function lemmas implies that, the potential function of the virtual graphs $VG(1|2,i)$ keep decreasing with increasing i. The **Convergence** lemma indicates that, when the potential function stops improving, the virtual graph $VG(final)$ is necessarily a union of successor-consistent and loopy cycles. In combination with the fact that all virtual graphs retain the weak connectivity property, this implies that the $VG(final)$ is either the correct virtual graph or a loopy cycle spanning all nodes. This concludes the proof. Conceptually invoking strong stabilization at this point, leads to the conversion of the loopy cycle to the ideal virtual graph. Even if the strong stabilization is involved alongside the convergence procedure, it can be shown that strong stabilization only improves the potential function, and thus the convergence is not slowed down by this process. Note that, in fact, a single invocation of the PR phase and possibly multiple invocations of the VR phase at every node is sufficient for correctness, but because each node operates asynchronously, it has to necessarily repeat the PR phase along with the VR phase every time.

At a single node, each invocation of the PR phase takes $O(N)$ time and $O(N^2)$ messages (since the routing may traverse all the nodes in the worst case, assuming no fingers). Each linearization phase takes $O(N)$ time and $O(N^2)$ messages (since at most N Succ Setup / Succ Confirm messages will be sent, each of them taking at most N hops). Thus, the total count for the system per iteration of both linearization and PR phase is $O(N)$ time and $O(N^3)$ messages. Combining this with the upper-bound of $O(N^3)$ on the potential function, since the ending potential function is at least zero, the total number of iterations is $O(N^3)$. Combining the number of iterations with the time and overhead per iteration, we get the upper-bound on the convergence time as $O(N^4)$ and on the message complexity as $O(N^6)$. Note that strong stabilization may take at most $O(N^3)$ time and $O(N^3)$ overhead per node [7], so that term can be absorbed into $O(N^4)$ time and $O(N^6)$ messages. We will now prove each of these lemmas, and thus the main theorem is proved.

LEMMA 1. **PR Connectivity:** *If the physical graph G is connected, then the virtual graph $VG(1,1)$, produced at the end of the PR phase in the first iteration, is weakly connected.*

PROOF. Consider any edge in the physical graph between two nodes m, n. Then, we will show there is a sequence of edges (ignoring direction) from m to n in $VG(1,1)$. To see this, consider the edge (m,n) and the Routing messages sent from, say, m to n during this PR phase in order to find a successor. Let the Routing Reply message from n have the source-route $n, n_1, n_2, \cdots, n_a$ where n_a is the prospective successor and $n_1, \cdots$ are the nodes on the *physical* path. Then, n_a is added to the list of candidate successors, and thus m will send a Succ Setup message to n_a. Thus, the state at m, $Succ(m)$, is updated with n_a, which implies that there is an edge from m to n_a in the virtual graph $VG(1,1)$. Since there is a sequence of virtual edges from n_1 to n_a and a newly added edge from m to n_a, ignoring the direction of these edges, there is a path from m to n. Note that n_a can be equal to n, in which case, the source-route is empty, and proof still holds.

Since this holds for any edge (m,n) in the original graph G, the fact the graph G is connected and the fact that an edge (m,n) corresponds to a path (ignoring edge orientation) from m to n in $VG(1,1)$, together imply that $VG(1,1)$ is weakly connected. □

LEMMA 2. **VR Phase Retains Connectivity:** *If the virtual graph $VG(1,i)$ is weakly connected, then the virtual graph $VG(2,i)$ is weakly connected.*

PROOF. This fact is easy to see from the nature of the linearization phase (illustrated in Figure 2). Notice that every edge in the virtual graph between the linearizing node and any of its prospective successors or predecessors is replaced by either a path or retained as a edge. The edge from the node x to the chosen successor s_1 and the chosen predecessor p_a are retained. The edge from x to s_i $(1 < i \leq b)$ is replaced by a path $x, s_1, \cdots, s_{i-1}$. Similarly, the edge from p_i to x is replaced by the path $p_i, p_{i+1}, \cdots, p_a, x$. Thus, if the virtual graph $VG(1,i)$ is weakly connected, upon the completion of any number of invocations of the VR phase, it will continue to be weakly connected. □

LEMMA 3. **PR Phase Retains Connectivity:** *If the virtual graph $VG(2,i)$ is weakly connected, then the virtual graph $VG(1,i+1)$ is weakly connected.*

PROOF. The PR phase only adds prospective successors to the virtual graph; it does not remove any existing ones. And since, we are analyzing the time duration where there is no churn, the current successor and the source-route to it remain valid. Therefore, this fact is trivially true. □

LEMMA 4. **VR Phase always decreases potential function:** *Provided $VG(1,i)$ has any node with indegree or outdegree exceeding 1, then $d(VG(2,i)) < d(VG(1,i))$.*

PROOF. Consider the actions performed by node n in the linearization phase. It has a number of outgoing successor pointers $s_1, \cdots, s_b$ and a number of incoming successor pointers $p_1, \cdots, p_a$. Consider any node n with in-degree/out-degree exceeding 1. Both cases are symmetric, so let us consider the case when the out-degree of n exceeds 1. In this case, a pointer from n to s_j in $VG(1,i)$ is replaced by a pointer from s_{j-1} to s_j in $VG(2,i)$ (for $j > 1$). The contribution by the old pointer to $d(VG(1,i))$ is $d(n, s_j)$, while the contribution by the new pointer to $d(VG(2,i))$ is $d(s_{j-1}, s_j)$. Notice that since $s_{j-1} \in [n, s_j]$, $d(n, s_{j-1}) + d(s_{j-1}, s_j) = d(n, s_j)$. Further, since $n \neq s_{j-1}$, $d(n, s_{j-1}) > 0$, which implies that $d(s_{j-1}, s_j) < d(n, s_j)$. Thus, expressing $d(VG(1,i))$ and $d(VG(2,i))$ as sum over pointers, it is clear that $d(VG(2,i)) \leq d(VG(1,i))$ since the contribution by each individual pointer either decreases or remains the same. Further, since there is at least one pointer whose contribution decreases (since there is at least one node with in-degree or out-degree exceeding 1), $d(VG(2,i) < d(VG(1,i))$. □

LEMMA 5. **PR + VR Phase decrease potential function:** $d(VG(2,i+1)) \leq d(VG(2,i))$.

PROOF. First, observe that the PR phase might add additional successors to the state $Succ(n)$ at node n. If these new successors help node x gain a closer successor, then the node potential of n decreases. Otherwise, the VR phase will reject those successors by sending Succ Setup messages on behalf of other nodes. Thus, even if the potential function increases due to the PR phase, (as it does due to the addition of a new node to $Succ(n)$), the distance metric will still suffer a overall drop due to the VR phase. This argument is the same as in the previous lemma. □.

LEMMA 6. **Convergence:** *If $VG(2,i)$ does not have any node with indegree or outdegree exceeding 1, then it must be a union of successor-consistent or loopy cycles.*

PROOF. If $VG(2,i)$ does not have any node with indegree/outdegree exceeding 1, then it is clear that VG_i is a union of node-disjoint cycles $C_1, C_2, \cdots$ (to see this, simply follow pointers from any node n, it has to terminate at n eventually due to the fact that the in-degree and out-degree of every node is 1). If each node n locally ensures that it lies between its successor s and predecessor p, then it has to be the case that each cycle C_i is either the successor-consistent cycle or a loopy graph. □

3.3.6 Evaluation

We now present some empirical results on maintenance overhead and convergence time of our maintenance algorithm. We ignore stretch because it has been well studied in VRR and ROFL: with path caches, VRR reports a stretch of 1.57 in the wireless context, and ROFL reports 2.5 in the Internet context. The maintenance overhead and convergence time is shown under two kinds of simulations. In the first kind, we allow a network of N nodes to boot up one after another, and we let it converge correctly. Then, we allow a single node to join/leave this network. We then measure the amount of time units elapsed and the number of messages exchanged until convergence. Our second kind of simulations is a stress test, where after the network of N nodes has converged, we induce a period of high stress where a fraction α of the current nodes leave and an equal number of new nodes join simultaneously (in our simulation) giving the protocol no time to adjust. Then, we measure the overhead and convergence time as a function of the stress parameter α. Also, due to space constraints, we only present results due to node failures here; results due to link failures present similar trends. We present a summary of our results here; for more details, please refer our technical report (not presented due to anonymity constraints).

Setup: We experimented with networks of size ranging from 20 nodes to 600 nodes placed in a grid, and the links are induced by the disk model of radio connectivity. The average degree of the network of 20 nodes is 6, and this linearly scales up as the number of nodes increase (since we place them in the same area). In calculating the convergence time, we assume that a message takes one time unit to traverse a link. We present the convergence overhead in terms of the number of messages (we detect convergence by checking if the virtual graph matches the correct one).

Single node joins and leaves: The control overhead and single node join varies between 13 messages for a 20 node network to about 80 messages for a 600 node network. The convergence time varies between 8 and 39 over the same range of network sizes. We not discuss results for failures because they are similar. The join overhead and convergence time both grow slowly (much slower than linear) with the size for the network, thus reflecting that the theoretical analysis is far too conservative.

Stress test: We now describe the control overhead and the convergence time for the stress test in a 600 node network. The convergence time remains nearly constant (between 24 and 25 units) reflecting that the diameter of the network is short enough to allow fast convergence. The message overhead, on the other hand, increases linearly since as the fraction of stressed nodes increases, the number of nodes rejoining also increases. The average message overhead per node is about 30 messages per node at a stress parameter of 0.1, while it reaches about 400 messages per node at 0.5 stress. The convergence time is between 20 and 24 seconds reflecting that the number of rounds of the VR phase is nearly constant, but that the number of messages sent per invocation is increasing rapidly. We have not explored optimizations to the VR phase in detail, but it is possible to envision simple optimizations that may help, such as re-concile one successor after another, instead of all at once.

4. APPLICATION TO OVERLAY DHTS

We now draw an unexpected connection between the eventual consistency of DHT-based routing and the eventual consistency of overlay-level DHTs. It is been well-noted [14] in the research literature that building DHTs on Planetlab is hard because of non-transitivity connectivity issues, transient connectivity, and heavily-loaded nodes. While the last issue is specific to Planetlab, it is believed that the issues of non-transitive connectivity and transient connectivity may be more pervasive and symptomatic of the general Internet.

Such connectivity issues generally lead to problems in the maintenance of the structure of the DHT. For example, if A can talk to B and B can talk to C, but A cannot reach C, then A might consider B as its successor, while B may repeatedly nominate C as A's successor. This can occur when the identifiers of these nodes are distributed as [A,C,B] in the identifier space. A however would repeatedly attempt to gain B as its successor since it assumes that C is down. Thus, when different nodes have different notions of which nodes are alive, the inconsistency can lead to convergence issues. Further, transient connectivity problems typically lead to some induced churn in the DHT, and, in combination with the non-transitivity issues, the structure of the DHT may be disrupted, and in fact, may never converge to the correct structure. Even worse problems occur when the DHT nodes may be partitioned from one another, in which case even the set of DHT nodes in one partition may not be able to discover each other, and maintain a DHT amongst themselves.

At present, there is no clean solution, theoretical or practical, to this problem. In practical deployments of OpenDHT [13] and Pastry [8], the leaf sets run a link-state like protocol to maintain reachability in the face of non-transitivity. This solution does not guarantee consistency in the face of large-scale non-transitivity, when members of a leaf set may be able to reach other only through nodes not in this leaf set.

We now show how our decentralized maintenance algorithm for DHT-based routing applies in the overlay context, and thus helps us develop the first clean theoretically correct solution to this problem.

4.1 Leveraging our Maintenance Algorithm

One can model a overlay-level DHT as follows. One can consider it as a *complete* graph on the N DHT nodes. Thus, the adjacency matrix is typically all ones, provided the Internet is working properly. During periods of non-transitivity etc, this need not be true. Now, one can simply treat these overlay-level DHT participants as *nodes participating in a DHT-based routing protocol over the complete graph*. These nodes can run our maintenance algorithm in response to alterations in the connectivity of the Internet, which is perceived by the algorithm as the churn in links and nodes in the complete graph. Thus, even if the DHT is used for data storage, in order to ensure consistency, the overlay nodes choose random identifiers, and then participate in this DHT-based routing protocol so as to ensure the DHT remains consistent.

This solution is elegant in that it both achieves consistency in the face of Internet link failures, as well as, automatically doing overlay routing (similar to RON [29]), if and when the need arises. When the Internet offers perfect connectivity, the overlay-level neighbors can reach directly over IP. If this is not true, then the algorithm automatically discovers source routes and guarantees consistency. In our previous example, concerning the non-transitivity pattern of connectivity between A,B, and C, when A asks B in the maintenance algorithm to discover its successor, B would return the source-route via itself to C. Thus, A would use the source-route [B,C] in setting up a link to its successor C, and would successfully setup a bi-directional link with C by relaying through B. The properties of our solution (in terms of convergence time and overhead) are similar to those analyzed theoretically and empirically in the previous section, so we do not aim to provide such results here. Our previous experiments modeled extreme non-transitivity (since the graph is far from fully connected), and thus those experiments are, in fact, a serious stress test from the point of view of overlay DHTs. We now deal with some of the details of applying our maintenance scheme to overlay DHTs.

Knowledge of Physical Neighbors: We note that in order to implement our DHT-based routing maintenance algorithm, each node is required to know all its physical neighbors. This is trivially possible in, say, a wireless network, but in the wide-area Internet, this requires that each node have a exhaustive list of all nodes in the system. This list is allowed to include dead nodes, but no new node can join the system unless everyone's list includes that new node. This property is required so that all nodes have a consistent view of all nodes in the system; otherwise, the complete graph abstraction breaks down, and the consistency can no longer be guaranteed.

We believe that, although far from trivial, this property can be achieved if it is acceptable for joins of new nodes to be delayed until it can be ensured that all nodes have included the new node in their list. We also note that this condition is necessary for strong consistency. Otherwise, if the list of nodes on node X does not include all nodes in the system, then upon the failure of all nodes in this list and those nodes pointing to and from X, the node X would be disconnected from the rest of the DHT nodes, who themselves may still be connected. Thus, if one desires the strong consistency requirement that, "irrespective of the connectivity pattern of the Internet, if the graph induced by the DHT nodes is connected, then the DHT is required to function correctly", then each node necessarily has to know all nodes in the system. If this assumption is reasonable, as it may be for carefully run DHTs offering services on Planetlab and running on a few hundred to a thousand nodes, we believe our work may be useful in this context.

Overhead of the PR Phase: We note that the PR phase, as described in the maintenance algorithm in the previous section, requires periodic successor requests via all physical neighbors. In the context of overlay-level DHTs, this period can be set extremely high to ensure low overhead in practice. We simply require a background gossiping style process at every node that operates as follows: it chooses a random node from the list of all nodes in the system, and requests its successor through that node (directly over IP). If this returned successor does not match its current successor, then the background process invokes the VR phase, which performs the reconciliation; otherwise, this process repeats its actions after its chosen period. This period can be chosen to be fairly low in practice, since our only goal is to provide eventual consistency; any non-transitivity will be detected and healed eventually. Thus, we believe the overhead of the PR phase can be held fairly low in practice.

Overhead of Probing: We distinguish between the *nearly static state*, the list of all nodes maintained at every node, and the *dynamic state*, the list of overlay neighbors at every node. The former is updated only when new nodes join the system, and further, need only be stored in memory or on disk. The latter is the one that requires constant probing in order to ensure that the overlay neighbors are up. Thus, our requirement of this complete list of nodes at every node does not imply an increasing in probing overhead. Of course, any source-routes of multiple hops set up to deal with any non-transitivity in the Internet will require probing over each of these hops; this is clearly necessary to route over multiple hops, and further, this increasing in probing overhead is proportional to the degree of non-transitivity in the Internet. Also, note that any uni-directional links will be simply considered as a failed link in our algorithm, since for a link to be considered active, it should be possible for one side to receive an acknowledgment from the other. Thus, the presence of uni-directional links does not lead to any consistency issues with our algorithm; such links will simply go unused.

5. RENDEZVOUS AND CONVERGENCE

The convergence results in Section 3 show that the convergence time, although low in practice for reasonable-sized networks, is still higher than, say, link-state routing, where the convergence time is constrained only by the diameter of the network (and in practice, timers etc). This trade-off is in some sense fundamental; intuitively, the lesser state requirement in DHT-based routing requires a carefully maintained

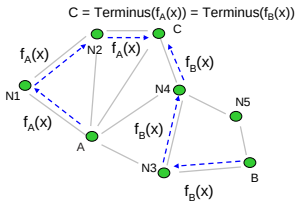

Figure 4: **Our Model of a Rendezvous Service (Grey solid lines refer to links, Dashed blue lines refer to the path taken by the rendezvous service computation)**

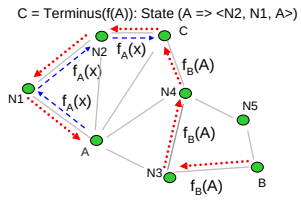

Figure 5: **Building Routing From Rendezvous (The Dotted red line reflects the path of the packet, The Blue dashed line and Grey solid lines reflect the rendezvous service computation and physical links respectively)**

structure, which clearly requires some time to converge.

In this section, we seek to develop new protocols that improve on the convergence time aspect of DHT-based routing at the expense of higher state requirements. Our methodology is to define the notion of a rendezvous service, a generalization of DHT-based routing, and then describe how to use a rendezvous service to build a routing service. We will thus expose the connection between a rendezvous service and a routing service. Of course, this exercise would be of academic interest if a DHT were the *only* way (or, at least, the only known way) to build a rendezvous service. This fortunately is not true; we go on to discuss other proposals in the literature (and even invent a new simple one) that may be viewed as a rendezvous service, and this suggests new routing protocols based on our prescription.

In the next section, we will discuss our general notion of a rendezvous service, and then describe the new routing protocols that we can obtain based on this notion.

5.1 Rendezvous Service

We now define a rendezvous service as follows. A rendezvous service is some distributed service (illustrated in Figure 4) built over a set of nodes $\mathbb{N}$ that allows them the following functionality: every node $n \in \mathbb{N}$ can compute a function $f_n : x \mapsto \mathbb{N}^*$ where x is any $k - bit$ value for some k (*i.e.,* $x \in [0, 2^k)$). The output of this function is a route y originating at n and terminating at some other node n'. It is represented by a sequence of node labels where, implicitly, there is a physical link between consecutive nodes on a source-route. We will use the notation $terminus(y)$ to represent the last node on the source-route y. Such a distributed service must satisfy the following requirements to be termed as a rendezvous service:

- The rendezvous service is responsible for updating the state used in the distributed computation of the function f_n, so that, when any node n computes the function $y = f_n(x)$ (for any x), after waiting for a sufficiently long time after any network churn events, the route y is guaranteed to terminate in an *alive* node and consist only of *live* links.
- When any two nodes n_1, n_2 compute their functions f_{n_1}, f_{n_2} for any value x, we require that the routes $y_1 = f_{n_1}(x), y_2 = f_{n_2}(x)$ both terminate at the same node (even if the exact nodes on the routes themselves are different). In other words, we require that for all nodes pairs n_1, n_2 and all possible x, $terminus(f_{n_1}(x)) = terminus(f_{n_2}(x))$.

The second criterion is the reason where we have chosen to call this service as a *rendezvous* service; it allows two nodes with a common "intent" x to reach the same node. Once again, this property need only be satisfied after a sufficiently long time after any failure event. Although the definition may seem complex, a very simple rendezvous scheme can be easily built on top of a routing scheme like path-vector routing.

Consider a path-vector routing scheme which allows every node to deduce the set of all alive and reachable nodes in the system (since the neighbors of a dead node would withdraw their advertisements). Further, each node would also know a source-route to every other alive and reachable node in the system. We now construct a simple rendezvous service $f_n(x)$ from this routing protocol. In order to compute the function $f_n(x)$, the node n first performs a (local) consistent hash (as defined by Karger *et al.* [5]) to compute a suitable terminus t. Since the set of all alive node labels is known locally, they can be hashed into a common bit-space along with the value x. Then the numerically closest node label to x, but not overshooting it, as per the consistent hashing definition, is chosen as the terminus t. After picking the terminus t, the node n then looks up its path vector table to pick a route to the terminus t, and this route represents $f_n(x)$. Clearly, provided the path-vector protocol converges in some finite time after any failure events, this satisfies both our requirements of a rendezvous service.

This description of building a rendezvous service from a regular routing protocol is merely an elucidatory excursion; it is hardly of any practical interest since our goal is to build routing protocols on top of rendezvous services, not the other way around. This is the direction we will pursue now.

Building Routing from Rendezvous: After having defined a rendezvous service, we discuss a generic technique (shown in Figure 5) to convert it to a routing scheme.

The first part of the idea is that each node A contacts its rendezvous location $terminus(f_A(A))$ periodically, and deposits state that enables its rendezvous location to reach it. Notice the self-referential computation: node A computes the function $f_A(A)$. We refer to $terminus(f_A(A))$ as the rendezvous location of A. This state is simply the *reversed* source-route: the rendezvous location $terminus(f_A(A))$ (in this case, C) is simply notified of the path corresponding to the reverse of $f_A(A)$. It maintains this state mapping from the identifier of A to this reverse source route.

The second part of the idea is that any node B can reach $terminus(f_A(A))$ by computing $f_B(A)$ itself; by our definition of a rendezvous service, this computation is guaranteed to give a route that terminates in $terminus(f_B(A)) = terminus(f_A(A))$. Thus, packets can be routed from B to A in two stages. First, B sends them to the rendezvous location of A by using the route $f_B(A)$. Second, the rendezvous

location of A retrieves the periodic state deposited by A, and now sends these packets using that state to A. Thus, the rendezvous location simply serves as a repository of state for a prescribed set of nodes, and this state is then used by any node in sending packets to this prescribed set of nodes.

We now formalize this scheme. Every node A periodically computes $f_A(A)$, and deposits the source-route corresponding to $f_A(A)$ at its rendezvous location $terminus(f(A))$. Any node B which wishes to sends packets to A computes $f_B(A)$ and sends them to the node $terminus(f(A))$. The latter is responsible for retrieving the source-route $f_A(A)$ from its stored state, and using them to send packets on to A.

We now wish to make a few finer points about this scheme. First, there is no need to perform the computation of the source-route, and then use it to send packets. The distributed computation can be simply tagged along with the packet, if desired. Second, although it appears that node A needs to update its state periodically, this is actually not the case if nodes on the path $f_A(A)$ are willing to maintain hard state, and inform the node n in the event of any link / node failure. Of course, it is always best to maintain a low periodic update in order to avoid the problems of hard state, but this rate can be very low. The fault in the route can even be discovered on-demand when a packet that needs to be sent using that route is dropped. Third, since the source route $f_A(A)$ needs to be reversed at the rendezvous location, we only expose bi-directional links to the rendezvous service. Finally, it might appear that triangular routing is a general problem with our prescription, but as shown in VRR [2], embedding pointer caches in the network allows one to "shortcut" these rendezvous points, significantly improving stretch and avoiding congestion at the rendezvous point. For example, when the packet for n is sent by some node to the rendezvous location for n, with careful selection of the number of rendezvous locations (since more than one location can be chosen for a given node), it is possible to engineer things so that, this packet is likely to hit a path cache containing a pointer to n, and thus, need not go to the rendezvous location at all. This is the approach taken in VRR.

DHTs and Rendezvous: The link between a DHTs and rendezvous service is now easy to see. To illustrate this, we will use Chord as the DHT algorithm due to its simplicity; and for now, we will consider Chord with only successor pointers.

The rendezvous location function $terminus(f_n(x))$ implemented by a DHT is the *consistent hashing* function, the same as in our earlier hypothetical example. Note, that since the terminus function does not depend on the node n computing the rendezvous function, we refer to it as $terminus(f(x))$. In the case of a DHT, the key x and node identifiers are hashed to a common space, and then if the key x falls between node identifiers, $[n_1 n_2]$ (appropriately defined for circular wrapping), then we consider n_1 to be $terminus(f_n(x))$. In Chord's terminology, thus, $terminus(f_n(n))$ would thus refer to the predecessor of n. This is the node that is supposed to constantly maintain a pointer to n.

The analogy to the rendezvous based prescription routing scheme is clear; the node n is the one responsible for notifying its predecessor when joining the network, and the predecessor of n is the one who volunteers to maintain the source-route to the node n. Thus, rendezvous function $f_n(n')$ is then simply the path of a lookup from the node n for the node n' which, after following the pointers along the lookup, terminates at the predecessor of n. Packets are sent to n along the path of the lookup, and then pick up the source-route deposited at the predecessor of n, and finally reach n. This description can be easily generalized to accommodate the complete Chord algorithm which includes multiple fingers (one can think of the multiple nodes pointing to n as the multiple rendezvous locations for n).

It is clear that our rendezvous service notion does not captures all the aspects of DHT-based routing. In particular, DHTs have two other notions which have no counterpart in our definition of a rendezvous service: the equal distribution of load among all nodes and the ability to bootstrap construction of these pointers using the DHT itself. Of course, it is not our intention to capture DHT-based routing precisely; our rendezvous service is supposed to be a generalization inspired by DHT-based routing, and if it was specified too precisely so as to completely model DHTs in every aspect, it would be hard to find other algorithms that provide the same service. As such, our definition of a rendezvous service is lax enough to allow other interesting implementations; this is the subject of the next section.

5.2 Building Rendezvous Routing Protocols

After we defined a rendezvous service as a generalization of DHT-based routing, we were surprised to find that we were able to cast two other proposals in literature as rendezvous services: Geographic Location Service [16] and Geographic Routing(*e.g.,* GPSR [17]). We will not discuss GLS here; please refer our technical report for more information. The high level point is that any location service (including GLS) can be seen as a rendezvous service, and thus, used to build a routing protocol constructed We now cast geographic routing as a rendezvous service which is a little more surprising.

Geographic Rendezvous Routing: The basic idea is that the rendezvous location function, $terminus(f_n(x))$, is defined as the node closest to the geographic location obtained by hashing n's identifier into the geographical coordinate space. The rendezvous function $f_n(x)$ is obtained by n by simply using the geographical routing algorithm to route to the hashed value of x and recording the source-route along this path if required. The GPSR [17] algorithm is guaranteed to terminate at the node closest to the geographical position at x. (of course, if there is a node at position x, GPSR will terminate there). Thus, a node n can periodically route to the geographical location corresponding to the hashed value of n, and the source-route accumulated on the way (or) its geographical position can be deposited at the rendezvous location. Any other node n' can route to the rendezvous location of n by using geographic routing to the hashed value of n, and then use the source-route or the geographical location to continue thereafter to n. Note that this proposal is somewhat similar to Geographical Hash Tables [18] and Beacon Vector Routing [19]; the main difference being that our protocol stores source-routes at the rendezvous point, instead of coordinates.

We refer to this routing method as *geographic rendezvous routing*, since we rely on the geographic system for rendezvous,

and then build a routing service out of this rendezvous service. The main advantage of geographic rendezvous routing is that, unlike geographic routing, it can route directly on node identifiers; no location service is required. This idea can be trivially extended to virtual coordinate routing (*e.g.,* NoGeo [26]) as well. The main advantage of geographic rendezvous routing in terms of consistency is in the following easily proved lemma.

LEMMA 7. *Consider a node n that updates its source-route at $f_n(n)$ at time T in geographic rendezvous routing. Consider a node n' that desires to send a packet to node n at time T' ($T' > T + D$ where D is the network diameter). If there are no intervening churn events between T and $T' + I$, where I is the in-transit time of the packet, then geographic rendezvous routing is guaranteed to deliver the packet.*

This lemma is easy to prove, and is based on the fact that geographic routing itself does not require any state, and its greedy routing procedure requires a node to know only the location of its neighbors. This guarantee is clearly much stronger than our guarantee for DHT-based routing; thus, in this case, our rendezvous based routing prescription has allowed us to gain faster convergence.

Beacon Rendezvous Routing: We now demonstrate the simplicity of our rendezvous primitive by illustrating a beaconing-based rendezvous service. The rendezvous location function, $terminus(f(x))$, that it implements is exactly the same consistent hashing function used in the DHT-based routing scheme; however, the difference is that instead of choosing from the entire set of nodes in the system, the consistent hashing function is applied only over a select set of *beacons*. The scheme works as follows.

A set of $O(\sqrt{(}N))$ beacons are chosen by some means; for example, each node can toss a coin and decide to be a beacon. These beacons use standard routing protocols (path-vector/distance-vector) to advertise themselves. Thus, every node can route to any of the beacons. The rendezvous function $f_n(x)$ is computed as follows: the node n locally determines the set of alive beacon nodes, uses consistent hashing to determine the beacon for $f_n(x)$, and uses the source-route to reach the beacon which serves as $terminus(f_n(x))$.

The beacon rendezvous routing protocol is extremely simple, and offers a state versus stretch trade-off that may be desirable in certain scenarios. The simplicity of this protocol also allows the following easily proven lemma.

LEMMA 8. *Consider a node n and its rendezvous location beacon $B(n)$. Let T be the time at which the beacon $B(n)$ (or some other node, on its behalf) last sent propagated a path-vector advertisement, and let T' be the most recent time at which time at which node n has updated its source-route at $B(n)$. Now, consider a node n' that wishes to send a packet to n at time T'' ($T > T + D, T > T' + D$ where D is the diameter of the network). Then, if there are no churn events on the path between $B(n)$ and n, n' in the duration $[min(T, T'), T'' + I]$, where D is the in-transit time of the packet, then the packet is guaranteed to be delivered.*

Thus, beacon rendezvous routing is much easier to analyze in terms of its convergence, and also boasts a much shorter convergence period as compared to DHT-based routing. This scheme has the advantage that every node maintains $O(D\sqrt{(}N))$ state (a path vector to all the beacons) which compares favorably to standard path vector schemes, but imposes a penalty on the beacons (here D is the diameter in terms of number of hops). We believe this penalty is acceptable since, in a regular routing protocol, such beacons would send these updates anyway. Of course, a different number of beacons can be chosen to obtain other trade-offs.

Note that it is possible to improve the robustness of the scheme by allocating multiple beacons to one node. Instead of choosing a single rendezvous node for one node, multiple hash functions (or) a modified consistency hash function could be used to map the identifier to a set of beacons. Each node is responsible for updating the source-route maintained at each of these multiple beacons, while a node wishing to send packets to another can send to any of the latter's rendezvous point. Further, with the addition of path caches, most traffic need never traverse any beacon.

Discussion: We believe that these two examples demonstrate that our definition of a rendezvous service, although inspired from DHT-based routing, allows new protocols with potentially much faster convergence time and other features lacking in DHT-based routing schemes. We hope that our identification of the rendezvous service as a more fundamental building block than DHT-based routing itself may pave the way for the development of new rendezvous services in the future, and perhaps newer routing protocols. It is as yet unclear whether any drastically new protocol will emerge from this abstraction; however, both the geographic rendezvous routing and the beacon rendezvous routing protocols are simple in construction and boast fast convergence. This connection between rendezvous service and routing appears novel and may be prove fruitful to explore.

6. RELATED WORK

We discuss related work into four categories. First, we discuss the research literature in Distributed Hash Tables (DHTs) and DHT-based routing. Second, we compare our algorithm for DHT-based routing maintenance with proposals for consistency in DHTs. Third, we list other recent for routing algorithms, and compare DHT-based routing with these proposals. Finally, we consider other rendezvous-based communication abstractions.

DHT Literature: Distributed Hash Tables [6–9] (DHTs) are highly scalable mechanisms for routing and object lookup. Traditional DHTs rely on an underlying routing protocol that provides point-to-point routing between overlay nodes. UIP [1] was the first to observe that a DHT itself could be used for routing and used to route between nodes in disparate address spaces. DHT-based routing was further explored in VRR [2] for wireless routing and for Internet Routing in ROFL [3].

One particularly challenging problem in DHT-based routing is in dealing with partitions and other inconsistencies. VRR [2] and ROFL [3] provided mechanisms for partition recovery. The schemes work by using a DSDV-like mechanism to discover and flood the identifier of the nodes closest to zero on the ring. These nodes are referred to as *beacons*. In particular, at each time step, each node looks at the identifiers it received through advertisements and its own identifier, selects the one closest to zero, and broadcasts that identifier

to its neighbors. This ensures that any node that should have the beacon as its successor can discover a path to the beacon. This approach works well for networks where there is a well-known stable node (small ad-hoc networks/sensornets and ISP networks), but is subject to poor performance in more decentralized networks where such a stable node cannot be found. Our work on eventual consistency includes a fully decentralized algorithm, and further, includes a proof of its eventual consistency.

DHT Consistency Literature: There has also been much work on maintaining ring consistency in traditional DHTs. ePost [27] detected partitions by periodically rejoining via a set of well-known bootstrap nodes. Chord [7] and Chen et. al. [28] used strong stabilization to detect and recover from loopy cycles. Our work on eventual consistency can be viewed as a generalization of these techniques to the case when DHTs are used to substitute IP routing.

Recent Routing Algorithms: The area of routing has seen considerable progress in the last 7-8 years. Geographic routing, or more generally, coordinate routing, has been been heavily researched in recent years. This area has progressed from the early work on geographical routing [17, 25] to virtual coordinate routing based on computed virtual coordinates [19,26]. The main limitation of these proposals is that a location service is required to translate from node identifiers to node coordinate. As our work points out, geographical routing itself may be used as a location service. There is even older literature in the theoretical community on compact routing [22,23] aimed at optimizing state trading off stretch. Most of these schemes are designed for the static case, and are difficult to extend directly to the dynamic case. A recent proposal for the dynamic case, based on the compact routing literature, is LAND [23], which however still relies of the seminal PRR structure [24] which is considered difficult to maintain correctly [8]. Thus, we believe that our DHT-based routing algorithms may be more amenable to the dynamic case, besides allowing a proof of eventual consistency.

Rendezvous-Based Communication Abstraction: Core-Based Trees (CBT [20]) and Internet Indirection Infrastructure (*i3* [21]) share some similarity with our notion of a rendezvous service, since in both case, they attempt to offer multicast or unicast functionality using rendezvous. The main difference is that, our definition is valid at the network layer and includes source-routes etc, whereas *i3* and publish-subscribe systems do not directly apply in the absence of underlying routing. We however believe CBT is closer to the spirit of rendezvous service since it is at the network layer; we plan to integrate the notion of multicast into our definition of rendezvous service in the future.

7. CONCLUSION

UIP, VRR, and ROFL have demonstrated that DHT-based routing presents an interesting and radical alternative that is technically superior to traditional routing algorithms in terms of state requirement and convergence overhead under most normal churn conditions. Yet, the fundamental principle behind DHT-based routing and its robustness remain unclear. Our work seeks to advance the theory of DHT-based routing by two major contributions: (b) by designing and proving the eventual consistency of a fully decentralized maintenance algorithm. (b) by generalizing DHT-based routing to the mechanism of rendezvous-based routing, and using the latter to develop routing protocols with faster convergence as compared to DHT-based routing. We hope that our work will inspire confidence in the DHT-based routing model and perhaps lead to newer instantiations of the rendezvous-based protocol. Although primarily an abstract exercise, we have pointed out several practical implications of our work throughout, ranging from the consistency of overlay DHTs to the requirement of location services for coordinate routing.

8. REFERENCES

[1] B. Ford, "Unmanaged Internet Protocol: taming the edge network management crisis, " CCR 34(1):93-98 (2004).
[2] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, A. Rowstron, "Virtual Ring Routing: Network routing inspired by DHTs", Sigcomm, September 2006.
[3] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, S. Shenker, "Routing on Flat Labels," Sigcomm, September 2006.
[4] Y. Afek, E. Gafni, M. Ricklin, "Upper and Lower Bounds for Routing Schemes in Dynamic Network", FOCS, '89.
[5] D. Karger, E. Lehman, F. T. Leighton, M. Levine, D. Lewin, R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", STOCS, 1997.
[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A Scalable Content-Addressable Network," SIGCOMM, 2001.
[7] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications", MIT-LCS-TR-819, 2002.
[8] A. Rowstron, P. Druschel, "Pastry: Scalable, Distributed Object Location And Routing For Large-Scale Peer-To-Peer Systems," IFIP Middleware, 2001.
[9] K. Hildrum, J. Kubiatowicz, S. Rao, B. Zhao, "Distributed Object Location in a Dynamic Network," SPAA, August 2002.
[10] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world", Proc. USITS, 2003
[11] N. Lynch, "Distributed Algorithms," Morgan Kaufmann Publishers.
[12] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT", In Proc. of USENIX Technical Conference, June 2004.
[13] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, H. Yu, "OpenDHT: A Public DHT Service and Its Uses," SIGCOMM, August 2005.
[14] Michael J. Freedman, Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica, "Non-Transitive Connectivity and DHTs", WORLDS, 2005.
[15] L. Subramanian, R. H. Katz, V. Roth, S. Shenker, I. Stoica, "Reliable Broadcast in Unknown Fixed-Identity Networks", PODC 2005.
[16] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing", Proc. Mobicom, August 2000.
[17] B. Karp and F.T Kung, "Greedy Perimeter Stateless Routing for Wireless Networks", MobiCom, Boston, MA, August, 2000.
[18] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage in SensorNets", In Proc. WSNA, 2002.
[19] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, I. Stoica, "Beacon-Vector Routing: Scalable Point-to-Point Routing in Wireless Sensor Networks", NSDI 2005.
[20] T. Ballardie, "Core Based Tree (CBT) multicast – architectural overview and specification." Internet Draft, October 1995.
[21] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, Sonesh Surana, "Internet Indirection Infrastructure," ACM SIGCOMM, August, 2002.
[22] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, M. Thorup, "Compact name-independent routing with minimum stretch", SPAA, 2004.
[23] I. Abraham , D. Malkhi , O. Dobzinski, "LAND: stretch $(1 + \epsilon)$ locality-aware networks for DHTs", Proc. Soda, 2004.
[24] C. Plaxton, R. Rajaram, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment", Proc. SPAA, 1997.
[25] P. Bose, P. Morin, I. Stojmenovi, J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks", Wireless Networks, Vol. 7, No. 6, 2001.
[26] Ananth Rao, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, "Geographic Routing without Location Information", MOBICOM, 2003.
[27] A. Mislove, A. Post, A. Haeberlen, P. Druschel, "Experiences in building and operating ePOST, a reliable peer-to-peer application," EuroSys, April 2006.
[28] W. Chen, X. Liu, "Enforcing routing consistency in structured peer-to-peer overlays: should we and could we?" IPTPS, February 2006.
[29] D. Andersen, "Resilient overlay networks", Master's thesis, Department of EECS, MIT, May 2001.
[30] L. Subramanian, M. Caesar, C. Ee, M. Handley, Z. Mao, S. Shenker, I. Stoica, "HLP: A next-generation interdomain routing protocol," SIGCOMM, 2005.
[31] Xiaowei Yang, David Clark, and Arthur Berger, "NIRA: A New Routing Architecture", IEEE/ACM ToN, 2007.
[32] D. Zhu, M. Gritter, and D. R. Cheriton, "Feedback Based Routing", HotNets-I, Sept. 2002.