# Scalable Routing on Flat Names

Ankit Singla and P. Brighten Godfrey
University of Illinois at Urbana-Champaign

Kevin Fall, Gianluca Iannaccone, and Sylvia Ratnasamy
Intel Labs Berkeley

## ABSTRACT

We introduce a protocol which routes on flat, location-independent identifiers with guaranteed scalability and low stretch. Our design builds on theoretical advances in the area of compact routing, and is the first to realize these guarantees in a dynamic distributed setting.

## 1. INTRODUCTION

Routing scalability is highly desirable for very large, dynamic, and resource-constrained networks, including many peer-to-peer systems and the Internet. Shortest-path routing algorithms (link state, distance vector, path vector, etc.) all [16] require $\Omega(n)$ memory at each router for a network with $n$ destinations, and at least as much communication and computation to build the routing tables.

One way to scale routing is to use a structured network topology that makes routing easy. Examples range from torus networks in supercomputers [25] to planar networks which permit greedy geographic routing [23] to hypercubes, small world networks, and other topologies in distributed hash tables [39,43,50]. But requiring a particular highly structured topology is not feasible for general-purpose networks.

For general networks, the common scaling technique is *hierarchy*: routing is performed over high-level aggregate units until it reaches the destination's unit, at which point routing proceeds at a finer granularity. For example, the Internet routes at the level of IP prefixes to a destination domain, and then over an intradomain routing protocol to a subnet. Hierarchy has two main problems. First, it can have arbitrarily high **stretch**, defined as the ratio of route length to the shortest path length. Simultaneously guaranteeing scalability and low stretch on arbitrary networks is a nontrivial problem which no deployed routing protocols achieve. Second, hierarchy requires **location-dependent addresses**, complicating management, mobility, and multihoming.

In response, numerous recent proposals suggest routing on location-independent **flat names** [5, 12, 13, 20, 21, 28, 42]. Flat names are a paradigm shift for the network layer: rather than requiring a location-dependent IP address to serve the needs of the routing protocol, a name is an arbitrary bit string that can serve the needs of the application layer. For example, a name could be a DNS name, a MAC address, or a secure self-certifying identifier [21,28,42]. But no previously proposed protocols exist for scalable, low-stretch routing on flat names.

This paper introduces a new routing protocol: Distributed Compact Routing, or "Disco". Disco builds on theoretical advances in centralized, static compact routing algorithms [3,44] and is the first *dynamic distributed* protocol to guarantee the following properties:

**Scalability:** A Disco router needs just $\tilde{O}(\sqrt{n})$ routing table entries[1] regardless of network topology.

**Low stretch:** Disco has worst-case stretch 7 on a flow's first packet, worst-case stretch 3 on subsequent packets, and much lower average stretch.

**Flat names:** Disco routes on arbitrary "flat" names rather than hierarchical addresses.

Disco's guarantee of $\tilde{O}(\sqrt{n})$ routing table *entries* translates to $\tilde{O}(r\sqrt{n})$ *bits* of state where $r$ is the size of a partial route; see discussion in §2.

Recently, several papers have introduced techniques which approach the above properties. In particular, citing the lack of a distributed compact routing protocol, VRR [9] introduced a novel DHT-inspired approach to route on flat names, but as we will see, VRR does not guarantee scalability and low stretch. S4 [34] is a distributed implementation of a compact routing algorithm of [44], but it does not route on flat names

[1]The standard notation $\tilde{O}(\cdot)$ hides polylogarithmic factors to aid readability. Disco actually has $O(\sqrt{n \log n})$ entries.


ACM CoNEXT 2010, November 30 – December 3 2010, Philadelphia, USA.

and as we will show, it breaks the state bound of [44] causing high per-node state on realistic topologies. Distributed compact routing was listed as an open problem by Gavoille [17].

We find, however, that guaranteed scalable, efficient routing on flat names is achievable. Disco is a careful synthesis of compact routing theory with well-understood systems techniques and a novel, low-overhead overlay network for disseminating routing state. That synthesis is the primary contribution of this paper. Disco thus represents a step towards closing the gap between theoretical and applied work on scalable routing.

The rest of this paper proceeds as follows. Sec. 2 discusses the need for our three requirements of scalability, low stretch, and flat names. We describe how past work has fallen short of these requirements in Sec. 3. Sec. 4 presents our protocol and Sec. 5 evaluates it. We conclude with a discussion of future work in Sec. 6.

## 2. REQUIREMENTS

This paper is guided by three key requirements.

**Guaranteed scalability:** The routing protocol should use little space and messaging regardless of the network topology. In particular, we wish to reduce state requirements from the $\Omega(n)$ bits at each node used by traditional routing protocols in an $n$-node network, to something asymptotically smaller, e.g., $\tilde{O}(\sqrt{n})$ per node. Guaranteeing scalability on any topology helps ensure that the protocol will continue to function smoothly despite future growth, higher dynamics, and new environments.

Disco meets this requirement for graphs in which we have a bound on the size of "explicit routes" within the local vicinity of each node. In particular, Disco has $\tilde{O}(\sqrt{n})$ routing table entries for a total of $\tilde{O}(r\sqrt{n})$ bits of state where $r$ is the maximum size of an explicit route, since these routes are embedded in nodes' addresses. (For example, we might have $r = O(\log n)$.) In §4.2, we discuss the need for this assumption and argue that it is reasonable. While in the worst case $r = \tilde{O}(\sqrt{n})$, in a router-level Internet map our addresses are shorter than IPv6 addresses.

**Guaranteed low stretch:** The protocol should find paths that are close to the shortest possible. Stretch is the ratio of the protocol's route length to the shortest path length. Since it is not possible to route on shortest paths using $o(n)$ state per node [16], some stretch is unavoidable given our scaling goal. However, we desire stretch to be bounded by a small constant in the worst case, so that local communication stays local and performance does not degrade regardless of the traffic demands.

**Flat names:** The protocol should route on arbitrary node names which may have no relationship with a node's location. In particular, this service should operate while preserving the stretch guarantee.

Flat names are widely recognized as a useful primitive. The location-independence of flat names aids mobility and eliminates the management burden of location-based address assignment. Numerous proposed redesigns of Internet routing use flat names to cleanly separate location from identity, including TRIAD [20], i3 [42], FARA [12], HIP [21], and LISP [13]. Flat names can also be self-certifying [28, 35, 46], where the name is a public key or a hash of a public key. This provides security without a public key infrastructure to associate keys with objects. Self-certifying names have been proposed for data objects in content-centric networks [28, 40], for a persistent and contention-free replacement for URLs [46], and for nodes in an accountable Internet architecture [5].

If low stretch were not a goal, supporting flat names would be straightforward. In particular, the routing system could resolve names into addresses—using DNS, a consistent hashing database over a set of well-known nodes [14, 26, 34], or a DHT [15]—and then route over addresses. But these solutions violate our stretch requirement: the resolution step might travel across the world even if the destination is next door.[2] Even though the resolution step may be needed for only the first packet of a flow, this latency may dominate for short flows; indeed, sub-second delays make a difference to users in web services such as Google [8]. Moreover, these solutions lack fate sharing [11]: a failure far from the source-destination path can disrupt communication. Similarly, an attacker far from the path could disrupt, redirect, or eavesdrop on communication.

Despite its desirability, scalable low-stretch routing on flat names has remained elusive. Indeed, satisfying all three requirements is an algorithmically challenging goal which remained essentially unsolved from the 1989 introduction of the problem [7] until 2003 [6], even in the static centralized case. And as we shall see in the next section, no *distributed* solution has been developed.

## 3. RELATED WORK

Scalable routing has a long and broad history, beginning with Kleinrock and Kamoun's 1977 analysis of hierarchical routing [27] and branching into many practical and theoretical application domains. We limit our discussion to routing protocols intended to scale for arbitrary topologies.

**Scalable routing in theory.** Universal compact routing schemes, beginning with early work by Peleg and Upfal [37], bound the size of routing tables and the

[2]Locality-aware DHTs rely on existence of underlying routing infrastructure and hence don't solve this problem. We discuss this issue in §3.

| Scheme | Scalable | Low stretch | Flat names |
|---|---|---|---|
| Shortest-path routing | | ✓ | ✓ |
| XL [31] | | ✓ | ✓ |
| Classic hierarchy | ✓ | | |
| Landmark routing [45] | | | |
| BVR [14] | ✓ | | |
| VRR [9] | | | |
| SEATTLE [26] | | ✓ | |
| S4 [34] | | ✓ | |
| Ford [15] | ✓ [a] | ✓ | |
| *Disco* | ✓ [a] | ✓ | ✓ |

[a]for graphs where the size of explicit routes is bounded (§2)

**Figure 1: Distributed routing protocols. *Scalable* indicates the protocol guarantees $o(n)$ state per node; *low stretch* indicates it has $O(1)$ or $O(\log n)$ stretch; *flat names* indicates it routes on flat names with low stretch.**

worst-case stretch on arbitrary undirected graphs. A series of results improved the state and stretch bounds, culminating in Thorup and Zwick [44] who obtained stretch 3 and space $\tilde{O}(\sqrt{n})$ per node, along with a family of schemes that have stretch $k$ and space $\tilde{O}(kn^{2/(k+1)})$ for any odd integer $k \geq 1$. This is essentially optimal, since stretch $< 3$ requires state $\Omega(n)$ [18], stretch $< 5$ requires state $\Omega(\sqrt{n})$, and in general, subject to a conjecture of Erdős, stretch $< k$ requires state $\Omega(n^{2/(k-1)})$ [44].

Those optimal results, however, are for the **name-dependent** model, where node names are chosen by the routing algorithm to encode location information—effectively, they are addresses. In the **name-independent** model, node names can be arbitrary ("flat") names. Abraham et al. [3] obtained stretch 3 with space $\tilde{O}(\sqrt{n})$, and Abraham, Gavoille, and Malkhi [2] obtained stretch $O(k)$ with space $\tilde{O}(n^{1/k} \log D)$ where $D$ is the normalized diameter of the network. Thus, surprisingly, the best name-independent schemes nearly match the name-dependent ones.

Unfortunately, this theoretical work on compact routing is far from practical for today's Internet. Most critically, it assumes centralized routing table construction and a static network.

**Scalable routing in practice.** Since compact routing algorithms are centralized and often relatively complex, it has not been clear how to translate them into practical distributed protocols; despite several attempts, results on the systems side are limited. Fig. 1 summarizes some of the protocols most closely related to Disco.

Landmark routing [45] has a similar motivation to our work: small space and stretch with a dynamic distributed protocol. However, landmark routing does not provide guarantees on either space or stretch for general topologies, and does not route on flat names.

SEATTLE [26] provides an Ethernet-like protocol while eliminating Ethernet's use of broadcast. It looks up Ethernet addresses in a consistent hashing database run on the routers, and routes along shortest paths after the first packet. It therefore does not route on flat names with low stretch. SEATTLE improves scalability relative to Ethernet, but still scales linearly in the number of routers $n$ (and resorts to an Internet-like hierarchy for large $n$, which would cause unbounded stretch).

XL [31] uses heuristics to significantly improve link state routing's messaging scalability, but does not improve the worst case, or routing table size.

Virtual Ring Routing (VRR) [9] and Routing on Flat Labels (ROFL) [10] route on flat names by applying techniques from DHTs to a physical network rather than in an overlay. However, these schemes have unbounded stretch on general topologies, and high stretch in practice; e.g., an average stretch of up to 8 in realistic topologies [10]. We confirm this in §5 and also show that VRR can have very high state for some nodes, even though it has low state on average.

Beacon Vector Routing (BVR) [14] represents a node address as a vector of distances to beacons and routes greedily with these vectors. While it greatly improves scalability, BVR can get stuck in local minima causing high stretch, and requires a name resolution step handled by the landmarks.

S4 [34] adapts one of the Thorup and Zwick [44] compact routing schemes to a distributed wireless setting. However, their adaptation breaks the per-node state bound of [44]; we show in §5 that S4 can indeed have high per-node state. Moreover, it does not route on flat names with low stretch.

Ford [15] evaluated a distributed version of [44] with $\Theta(\log n)$ stretch and state, but [15] does not route on flat names with low stretch. Also, Disco chooses a different point in the tradeoff space, with $O(1)$ stretch but $\tilde{O}(\sqrt{n})$ state. Both Disco and [15] assume a bound on the size of an explicit route (§2).

Westphal and Kempf [47] applied static compact routing algorithms to support mobility, but only with a fixed infrastructure and mobile leaf nodes.

Krioukov et al. [29,30] applied static compact routing algorithms to Internet-like topologies with promising results, but did not develop distributed protocols.

**Related techniques in overlay networks.** Distributed hash tables [39, 43, 50] may appear to satisfy our goals, as they route on flat names often with $O(\log n)$ routing table entries and within $O(\log n)$ overlay hops. However, DHTs have unbounded stretch: even a single overlay hop can be arbitrarily longer than the distance to the destination! More fundamentally, DHTs are not general-purpose routing protocols: they are overlay networks which depend on the availability of a general-purpose routing protocol beneath them.

Tulip [1] is an overlay network that routes with $\tilde{O}(\sqrt{n})$

state and round-trip[3] stretch 2 on flat names. Both our work and Tulip share theoretical techniques based on [3]. However, like DHTs, Tulip does not solve the network routing problem; its stretch guarantee effectively assumes a stretch-1 routing protocol beneath it. One could modify Tulip's data structures to store routes instead of addresses to achieve routing functionality. The problem then is constructing and maintaining this routing state in a *distributed* and *dynamic* fashion. Locality aware DHTs [4, 38] also suffer from this problem. It is this problem that Disco resolves.

**In summary,** to the best of our knowledge, the only proposed distributed protocol which is both scalable and has low stretch is [15], and no previously proposed practical protocol retains guarantees on scalability and stretch while routing on flat names.

# 4. DISTRIBUTED COMPACT ROUTING

This section begins with our assumptions and definitions (§4.1). We then present Disco's main components: a *name-dependent* distributed compact routing protocol, NDDisco (§4.2); a name resolution module (§4.3); and a distributed location database to achieve *name-independence* (§4.4). Finally, we prove Disco's state and stretch guarantees (§4.5). We defer an evaluation of messaging overhead to our simulations (§5).

## 4.1 Assumptions and definitions

We assume we are given an undirected connected network of $n$ nodes with arbitrary structure and link distances (i.e., link latencies or costs). We let $v \rightsquigarrow w$ denote a shortest path from $v$ to $w$ (in terms of distance, not hopcount), and let $d(v, w)$ denote the length of this path. The **name** of a node is an arbitrary bit string; i.e., a flat, location-independent name. We assume each node knows its own name and its neighbors' names, but nothing else. Like past schemes, our protocol designates some nodes as **landmarks**, whose function will be described later. We let $\ell_v$ denote the landmark closest to a node $v$.

An event occurs with high probability (**w.h.p.**) if it has probability $\geq 1 - n^{-c}$ for some constant $c \geq 1$.

We assume nodes can estimate $n$. While this could be done in many ways, we propose the use of synopsis diffusion [36]. SD requires only extremely lightweight, unstructured gossiping of small "synopses" with neighbors and produces robust, accurate estimates (e.g., within 10% on average using 256-byte synopses).

## 4.2 Name-dependent compact routing

Disco begins with a name-dependent distributed compact routing protocol, **NDDisco**. This protocol guarantees worst-case stretch 5 on the first packet of a flow, worst-case stretch 3 on subsequent packets, and $\tilde{O}(\sqrt{n})$ routing table entries per node. But it is name-dependent: we assume the source knows the destination's current address (defined below), as opposed to just its name. NDDisco is based on a centralized algorithm of [44]. In this subsection we describe the components of NDDisco, and then compare it with S4 [34], another distributed protocol based on [44].

[3]Note this is a different definition of stretch than the more common one-way definition that we use in this paper.

**Landmarks.** A **landmark** is a node to which all nodes know shortest paths. Landmarks will allow us to construct end-to-end routes of the form $s \rightsquigarrow \ell \rightsquigarrow t$. Each of the two segments will be precomputed by the routing protocol, and the full route will be close to the shortest if $\ell$ is close to $t$.

Landmarks are selected uniform-randomly by having each node decide locally and independently whether to become a landmark. Specifically, each node picks a random number $p$ uniform in $[0, 1]$, and decides to become a landmark if $p < \sqrt{(\log n)/n}$. Thus, the expected number of landmarks is $n \cdot \sqrt{(\log n)/n} = \sqrt{n \log n}$ and by a Chernoff bound, there will be $\Theta(\sqrt{n \log n})$ w.h.p.

Since $n$ can change, nodes will dynamically become, or cease to be, landmarks. To minimize churn in the set of landmarks, a node $v$ only flips its landmark status if $n$ has changed by at least a factor 2 since the last time $v$ changed its status. This amortizes the cost of landmark churn over the cost of a large number ($\Omega(n)$) of node joins or leaves.

**Vicinities.** Landmarks gave us a way for a source $s$ to approximately locate a destination $t$, but if $s$ and $t$ are close, it will be a poor approximation relative to the distance between them. To solve this problem, each node $v$ learns shortest paths to every node in its **vicinity** $V(v)$: the $\Theta(\sqrt{n \log n})$ nodes closest to $v$. These sizes ensure that each node has a landmark within its vicinity w.h.p., which is necessary for the stretch guarantee.

**Learning paths to landmarks and vicinities.** Nodes learn shortest paths to landmarks and vicinities via a single, standard path vector routing protocol. When learning paths, a route announcement is accepted into $v$'s routing table if and only if the route's destination is a landmark or one of the $\Theta(\sqrt{n \log n})$ closest nodes currently advertised to $v$. The entire routing table is then exported to $v$'s neighbors.

With these rules, the protocol will converge so that $v$ knows the landmarks and $V(v)$, for a total of $\Theta(\sqrt{n \log n})$ routing table entries w.h.p. We note however that the control plane requires $\Theta(\delta\sqrt{n \log n})$ state for a node with degree $\delta$, because path vector stores the full set of route advertisements received from each neighbor. This may be acceptable for low-degree graphs or if routers with higher degree are more well-provisioned. We can reduce control state to $\Theta(\sqrt{n \log n})$ by either forgetting the unused announcements as in Forgetful Routing [24],

or by preventing them from being sent by having each node inform its neighbors of the "radius" of its vicinity.

**Addresses.** The address of node $v$ is the identifier of its closest landmark $\ell_v$, paired with the necessary information to forward along $\ell_v \rightsquigarrow v$. Addresses are location-dependent, of course, but they are only used internally by the protocol, and are dynamically updated to reflect topology changes.

But what is "the necessary information to forward along $\ell_v \rightsquigarrow v$"? In the version of NDDisco evaluated here, it is an *explicit route* consisting of a list of labels, one for each hop along the $\ell_v \rightsquigarrow v$ path. A node's address is thus variable length with size dependent on the number of hops to its nearest landmark. The reader may notice that in the worst case this address size is quite large, as much as $\tilde{O}(\sqrt{n})$ bits in a ring network of $n$ nodes. This would be too large for a packet header and would explode our state bound since we will have to store many addresses on some nodes (§4.3). Our design decision to use explicit routes thus invites some explanation.

Most importantly, the explicit route is in practice extremely compact. Each hop at a node of degree $d$ is encoded in $O(\log d)$ bits following the format of [19]. We measured the size of explicit routes in CAIDA's router-level map of the Internet [48] by picking random landmarks and encoding shortest paths from each node to its closest landmark as a sequence of these $O(\log d)$-bit encodings of the node identifiers on the path. The maximum size of our addresses is just 10.625 bytes (less than an IPv6 address), the 95th percentile is 5 bytes, and the mean—the important metric for the per-node state bound—is 2.93 bytes (less than an IPv4 address).

The explicit route *could* be eliminated. Briefly, an address would be fixed at $O(\log n)$ bits; each landmark $\ell$ would dynamically partition this block of addresses among its neighbors in proportion to their number of descendants, and this would continue recursively down the shortest-path tree rooted at $\ell$, analogous to a hierarchical assignment of IP addresses. Since this would complicate the protocol and actually *increase* the mean address size in practice, we chose the simpler explicit route design.

**Routing.** A source $s$ can now send to a destination $t$ as follows. If $t$ is a landmark or $t \in V(s)$, then $s$ can route along a shortest path to $t$. Otherwise, it extracts $\ell_t$ from $t$'s address and routes along $s \rightsquigarrow \ell_t \rightsquigarrow t$. (Recall that for NDDisco, unlike Disco, we assume that $s$ knows $t$'s address.) This first packet of the flow has worst-case stretch 5, a fact which was shown in [44] for a centralized algorithm, and which applies to our protocol since it is essentially a distributed implementation of [44].

Subsequent packets can do better. Upon receipt of the message, $t$ determines whether $s \in V(t)$. If so, $t$ knows the shortest path even though $s$ didn't (note that $s \in V(t)$ does *not* imply $t \in V(s)$). In this case, $t$ informs $s$ of the path $s \rightsquigarrow t$, and all subsequent packets follow this path. This is again essentially equivalent to the protocol of [44] which [44] showed guarantees worst-case stretch 3.

**Shortcutting heuristics.** In S4 [34], if at any point the packet passes through a node which knows a direct path to $t$, then the direct path is followed. We refer to this as "To-Destination" shortcutting. We implement two further optimizations. First, we try both the forward and reverse routes $s \rightarrow t$ and $t \rightarrow s$, and use the shorter of these. When combined with To-Destination, we call this "No Path Knowledge" shortcutting.

Second, a more aggressive optimization is for every node along the route to inspect the route and see whether it knows a shorter path to any of the nodes along the route (via its vicinity routes) in either forward or reverse directions, in which case the route is shortened. We call this "Up-Down Stream" shortcutting. The latter optimization requires listing the global identifiers of every node along the path. This can be done on a single initial packet. This can also be combined with using the reverse route (referred to as "Path Knowledge" then). Our simulations will show that this "Path Knowledge" optimization significantly reduces stretch, but due to the added complexity, we exclude it from the evaluation of our core protocol. All results discussed subsequently use the "No Path Knowledge" optimization.

**Comparison with S4.** Both NDDisco and S4 [34] are distributed protocols based on [44]. Specifically, S4 is based on the algorithm in Sec. 3 of [44] where, rather than knowing vicinities as described above, each node $v$ knows its "cluster": nodes that are closer to $v$ than to their closest landmarks. To adapt [44] to a distributed setting, S4 selects uniform-random landmarks rather than using the multiple-pass algorithm of [44]. Unfortunately, this breaks the per-node state guarantee; in §5 we will see that S4's per-node state can be quite high. Briefly, the problem is that some nodes can be close to *many* nodes in the network, exploding their cluster size. (This is why [44] needed to select landmarks with a more complex algorithm.)

NDDisco avoids this problem by having each node $v$ store its vicinity (as defined above: the $\tilde{O}(\sqrt{n})$ nodes closest to $v$) rather than its cluster. This enforces a bound on the number of routing table entries at each node for any network, but does have two consequences. First, it requires the source to query the destination's vicinity in order to guarantee stretch 3, as described above; this design is essentially a distributed version of the "handshaking-based" scheme of Thorup and Zwick (Sec. 4 of [44]).

Second, this design leads to our use of explicit routes

in addresses. S4 ensures that for any node $v$ and its closest landmark $\ell_v$, node $v$ will appear in $\ell_v$'s cluster. But in NDDisco, it is possible that $v \notin V(\ell_v)$. Thus, we include the explicit route $\ell_v \rightsquigarrow v$ in $v$'s address, so packets sent to $\ell_v$ can be forwarded to $v$.

S4 and NDDisco do not route on flat names. In the rest of this section, we build Disco, which adds routing on flat names on top of NDDisco.

## 4.3 Name resolution

NDDisco (§4.2) requires a sender to know the destination's current address. We can solve this by running a consistent hashing [22] database over the (globally-known) set of landmarks, similar to [14,34]. Every node is aware of its own address ($\ell_v, \ell_v \rightsquigarrow v$), so it can insert it into the database, and other nodes can query the database to determine $v$'s address. This state is soft: it can be updated, for example, every $t$ minutes and timed out after $2t + 1$ minutes. In our simulator, $t = 10$. We will show (§4.5) that this adds $O(\sqrt{n \log n})$ entries to each landmark, so the state bound is preserved.

However, this is only a partial solution, because the first packet of a flow may have arbitrarily high stretch. As discussed in Sec. 2, this impacts latency, reliability, and security. We will, however, use this name-resolution database as a component of Disco in the next section.

## 4.4 Name-independent compact routing

Disco is comprised of NDDisco (§4.2), name resolution (§4.3), and a distributed name database. The last step ensures constant stretch while routing on flat names, and is what we describe in this section.

**Overview.** We build a distributed database which maps nodes' names to their addresses. The database requires the key properties that (1) any necessary query can be accomplished with low stretch, (2) the $\tilde{O}(\sqrt{n})$ per-node state bound is not violated, and (3) maintaining the state in the face of network dynamics requires little messaging. We adopt the idea of "color"-grouping of nodes from [2], but adapted with sloppy grouping and routing schemes which are more amenable to a distributed setting. We then design a random overlay network organized around these groups which can efficiently distribute flat-name routing state.

**State.** We begin with a well-known hash function $h(v)$ (e.g., SHA-2) which maps the node name to a roughly uniformly-distributed string of $\Theta(\log n)$ bits. Node $v$ is a member of a "sloppy group" of nodes that have in common the first few bits of $h(v)$. Specifically, let $G(v)$ be set of nodes $w$ for which the first $k := \lfloor \log_2(\sqrt{n/\log n}) \rfloor$ bits of $h(w)$ match those of $h(v)$. Thus, $G(v)$ will contain $O(\sqrt{n \log n})$ nodes w.h.p. The group is sloppy because this definition depends on $v$'s estimate of $n$, which will vary slightly across nodes. Node $v$ then ensures that every node in $G(v)$ stores $v$'s address. We will return to how exactly this is done shortly.

This definition of the sloppy group $G(v)$ has two important properties. First, it is consistent in the sense of consistent hashing [22]: a small change in the number of nodes does not result in a large change in the grouping. This is essential to scalably handle dynamics. More precisely, there will be no change in $k$ and hence no change in the grouping unless $n$ changes by a constant factor.[4]

The second important property is that if the grouping *does* change, it corresponds to splitting a group in half or merging two groups. This means that nodes that have slightly different opinions about the value of $n$ will still roughly agree on the grouping, which helps us handle (constant-factor) error in estimating $n$.

We show how to maintain the sloppy group state shortly; but first, we describe how to use it.

**Routing.** To route from $s$ to $t$, node $s$ first checks (as in NDDisco) whether it knows a direct path to $t$, either because $t$ is a landmark or $t \in V(s)$. If so, it routes directly.

Otherwise, if $s$ knows $t$'s address (i.e., because $s \in G(t)$), it can route to $t$ according to NDDisco.

Otherwise, $s$ locally computes $h(t)$. It then examines its vicinity and finds the node $w \in V(s)$ which has the *longest prefix match* between $h(w)$ and $h(t)$. (This can be optimized slightly to be the closest node $w$ with a "long enough" prefix match.) Due to our choice of $k$, with high probability, $w \in G(t)$, so $w$ will know $t$'s current address ($\ell_t, \ell_t \rightsquigarrow t$). The full path (if no shortcutting occurs) is thus $s \rightsquigarrow w \rightsquigarrow \ell_t \rightsquigarrow t$. We show that this has stretch $\leq 7$ in §4.5.

After the first packet, $s$ knows $t$'s address, so routing proceeds as in NDDisco with stretch $\leq 3$.

Note that there is a vanishingly small but nonzero probability that $t$'s address is not found because our state is maintained correctly only with high probability. Such events never occurred in our simulations. However, if this did occur routing could operate correctly by simply using name resolution on the landmark database (§4.3) as a fallback.

**Sloppy group maintenance.** We now describe how a node $v$ can ensure that the nodes $G(v)$ have $v$'s address. (Note that $v$ does not know which nodes these are.)

A naïve solution is to use the consistent hashing-based name resolution database running on the landmarks (§4.2). Each node $v$ already stores its name and address at the landmark which owns the key $h(v)$. It is easy to see that all of $G(v)$ will be stored on a predictable set of $O(\log n)$ landmarks, from which any node

[4] If $n$ is such that $k$ is near the boundary of two values, we can avoid the frequent "flapping" that could result by changing the sloppy group only when the estimate of $n$ changes by at least some constant factor (e.g., 10%).

can therefore download its group membership information. This, however, imposes a high messaging burden on the landmarks: if every node changes its address once per minute, the landmark would have to relay $\tilde{O}(\sqrt{n})$ addresses to each of $\tilde{O}(\sqrt{n})$ nodes for a total of $\tilde{O}(n)$ bytes per minute.

Disco instead adopts a more decentralized solution. Each node $v$ maintains a set of **overlay neighbors** $N(v)$. Similar to a DHT structure, $N(v)$ includes $v$'s successor and predecessor in the circular ordering of nodes according to their hash values $h(\cdot)$. $N(v)$ also includes a small number of long-distance links called "fingers".

To select a finger, a node $v$ picks a random hash-value $a$ from the part of hash-space that falls within $G(v)$. Following [32], $a$ is picked such that the likelihood of picking a value is inversely proportional to its distance in hash-space from $h(v)$. Based on these rules, a node finds the name and address of a finger by querying the landmark-based resolution database for the node with the closest hash-value to $a$. In this manner, node $v$ picks a small constant number of fingers (we will test 1 and 3 in our simulations) refreshing the set $N(v)$ periodically at a low rate. It then opens and maintains TCP connections to each of these nodes, for an average of $|N(v)| \approx 4$ or 8 overlay connections (for 1 or 3 fingers respectively) counting both outgoing and incoming connections.

Within this overlay, we can efficiently disseminate routing state in a manner very close to a distance vector (DV) routing protocol. We describe the protocol through its differences from the standard DV protocol. First, we emphasize that we use this protocol only to *propagate address information*, rather than to find routes. Second, since we are only interested in disseminating addresses rather than finding short paths, we need not include a distance in the announcement, but we (obviously) must include the originating node's name and address. Third, nodes only propagate advertisements to and from nodes they believe belong to their own group, thus keeping address information local to each group. Fourth and most importantly, node $v$ propagates advertisements only to those nodes in $N(v) \cap G(v)$ which would cause the message to continue in the same direction: that is, announcements received from an overlay neighbor with higher hash-value are propagated only to neighbors with lower hash-values, and vice-versa. This eliminates distance vector's count-to-infinity problem because as announcements are propagated, their distance (in hash space) from the origin of the announcement strictly increases. Other aspects of the protocol (incremental updates, state maintained, etc.) are similar to DV.

Why does this design work? First note that although nodes differ in their opinion on the sloppy grouping, they don't differ substantially. In particular, since we can ensure that estimates of $n$ are within a factor of 2 of the correct value w.h.p., nodes will differ by at most *one bit* in the number of bits $k$ that they match to determine the grouping. Thus, there will be a "core group" $G'(v)$ such that all nodes in $G'(v)$ will agree that they are in the same group. $G'(v)$ is clearly connected via successor/predecessor links, so $v$'s announcement will reach all the nodes in $G'(v)$. Since $|G'(v)| = \Theta(\sqrt{n \log n})$ a Chernoff bound can be used to show that every node $s$ will have some member of $G'(v)$ in its vicinity w.h.p. The routing protocol above will then find such a node with its prefix-matching step, so $s$ will be able to route to $v$.

Beyond *correctness*, the design is *efficient* in two ways. First, the overlay has constant average degree, so each node will receive only a few copies of each announcement message. Second, these messages are propagated relatively quickly, as the expected path length in the Symphony topology is $O(\log^2 n)$[5] [32].

We briefly clarify two minor points. First, there might be conflicting announcements received by $v$ for some node $x$'s address; in this case, $v$ can simply use the announcement which was received from the node whose hash-value is closest to $h(x)$. Second, during convergence and maintenance of the overlay, $v$ may have all its announcements for some node $x$ temporarily withdrawn even though $x$ is still live. To provide reliable service during these periods, $v$ can delay removal of address state until some short period (e.g., 30 sec) has passed.

## 4.5 Guarantees

We next prove that Disco maintains low stretch and state. We do not bound the messaging overhead analytically, but we will simulate it in §5. Due to space constraints, we defer all proofs to [41].

**Stretch.** As previously noted, the results of [44] apply to our name dependent protocol NDDisco: it has stretch $\leq 5$ for the first packet of a flow, and $\leq 3$ for subsequent packets.

Our name-independent routing, however, lengthens paths further. We now show that it still maintains stretch $\leq 7$ for the first packet.

THEOREM 1. *After converging, Disco routes the first packet of each flow with stretch $\leq 7$, and subsequent packets with stretch 3 w.h.p.*

**State.** For convenience, we analyze state in terms of the number of entries in the protocol's routing tables. Each entry may contain a node name and address, which

[5]Since we are gossiping on many paths rather than routing along a greedy path, we conjecture that this bound can be improved to $O(\log n)$ along the lines of [33].

in turn contains a landmark name and an explicit route from the landmark to the destination. (Recall from §4.1 that the explicit route will in practice occupy just a few bytes, and for extreme cases a variant of our design can ensure $O(\log n)$-size addresses for any topology.)

THEOREM 2. *After converging, with high probability, each Disco node of degree $\delta$ maintains $O(\delta\sqrt{n \log n})$ entries in its routing tables including the control and data planes, or $O(\sqrt{n \log n})$ using forgetful routing.*

# 5. EVALUATION

We evaluate Disco in comparison with two recent proposals for scalable routing, S4 and VRR, over a variety of networks. Our results confirm our main goals. First, we find Disco has an extremely balanced distribution of state across nodes, and across topologies, while in both S4 and VRR some nodes have a very large amount of routing state in realistic topologies. Second, Disco maintains low stretch in all cases, even for the first packet of a flow, while the other protocols can have high first-packet stretch; this is particularly evident for a topology annotated with link latencies rather than simply hopcount.

## 5.1 Methodology

**Protocols.** We evaluate five protocols: Disco, NDDisco, S4 [34], VRR [9], and path vector routing. NDDisco is our name-dependent protocol coupled with the landmark-based name resolution database; it is directly comparable to S4 in its goals but guarantees $\tilde{O}(\sqrt{n})$ per-node state. Our implementation of S4 is as in [34] except that we use path vector for cluster and landmark routing, making it more comparable to NDDisco. We evaluated VRR with $r = 4$ virtual neighbors as in [9]. VRR's converged state depends on the order of node joins; we start with a random node and grow the connected component of joined nodes outward.

**Simulators.** We simulated Disco, S4, and path vector in a custom discrete event simulator. For topologies larger than 1024 nodes, we built a static simulator which calculates the post-convergence state of the network. We evaluate VRR in the static simulator only. Our simulator for VRR scales poorly so we present only results on 1024 node topologies for VRR.

In many cases, for large topologies, we sample a fraction of nodes or source-destination pairs to compute state, stretch, and congestion.

**Topologies.** Our results include (1) a 30,610-node AS-level map of the Internet [49]; (2) a 192,244-node router-level map of the Internet [48]; (3) $G(n, m)$ random graphs of various sizes, i.e., $n$ nodes with $m$ uniform-random edges, with $m$ set so that the average degree is 8; and (4) geometric random graphs of various sizes with average degree 8.

## 5.2 Results

**State.** We measure data plane state for the protocols. This includes everything necessary to forward a packet after the protocol has converged: forwarding entries for landmarks and vicinities, name resolution entries on the landmark database, forwarding label mappings for our compact source route format in NDDisco, and the address mappings for Disco.

Fig. 2 shows S4 does well on the random graphs, but is extremely unbalanced on the Internet topologies. Intuitively, S4 does poorly on topologies where some nodes are more "central" than others. In fact, it is easy to show that S4 will have $\tilde{\Theta}(n)$ state on some nodes in the worst case; see the full version of this paper [41]. This demonstrates that S4's simplification of one of the algorithms of [44] can indeed cause high state.

In contrast, Disco and NDDisco have very balanced distributions of state in all cases. Note that NDDisco is a fairer comparison with S4 since both protocols are name-dependent, while Disco adds additional state for name-independence. Average state is slightly higher in NDDisco than S4 because of a differing design decision about vicinity size: S4 expands its cluster until it reaches a landmark, while NDDisco and Disco have vicinities which are fixed at $\Theta(\sqrt{n \log n})$ nodes. This difference is not fundamental to NDDisco, but is necessary for Disco so that there will be an intersection between each node's vicinity and each destination's sloppy group.

As Fig. 4 (left) and Fig. 5 (left) show, VRR fares very poorly compared to both S4 and Disco on both topologies in terms of state. It does even worse than path vector for a few nodes. This is because VRR constructs end-to-end paths and stores state at each node along the path, so in theory it could have as many as $\Theta(n^2)$ routing entries at a node (though it does not approach this worst case).

All results on routing state discussed so far only count the number of entries. In Table. 7, we present numbers for state in terms of kilobytes of memory. The size of source routes is determined using the scheme described in §4.2. As the table shows, the conclusions are similar when measuring bytes instead of entries.

**Stretch.** Fig. 3 shows the distribution of stretch in S4, Disco, and NDDisco. We call the reader's attention to the difference between the three graphs. In the Internet router and AS topologies, links are unweighted (or equivalently, all link latencies are 1). Thus, maximum stretch is limited simply because the ratio of the longest to the shortest path is bounded. In contrast, the geometric random graph includes link latencies and $S4$ experiences worst-case stretch of 72 while Disco's highest stretch is just over 2. Unfortunately, a latency-annotated Internet topology was not available to us; it

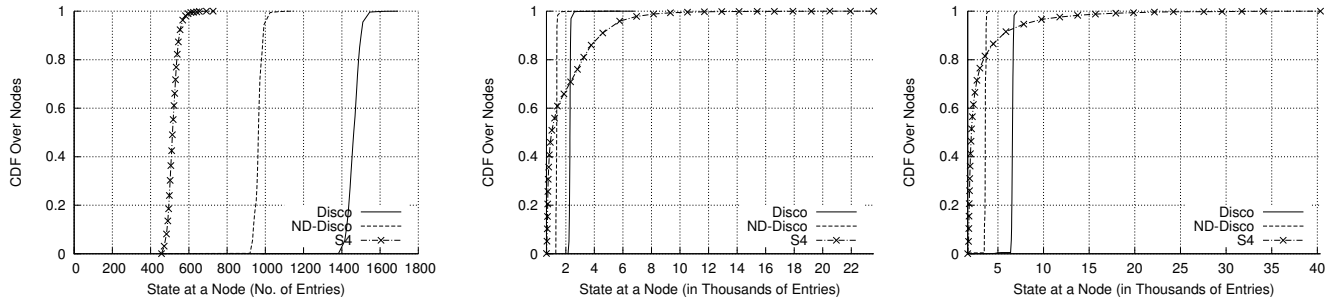



**Figure 2: State in a 16,384-node Geometric Random Graph (left), Internet AS-level graph (middle) and Internet Router-level graph (right).**

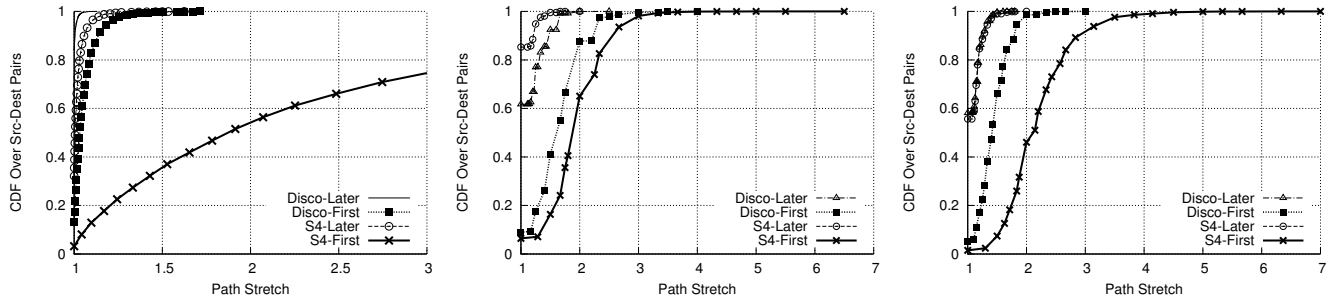



**Figure 3: Stretch in a 16,384-node Geometric Random Graph (left), Internet AS-level graph (middle) and Internet Router-level graph (right).**

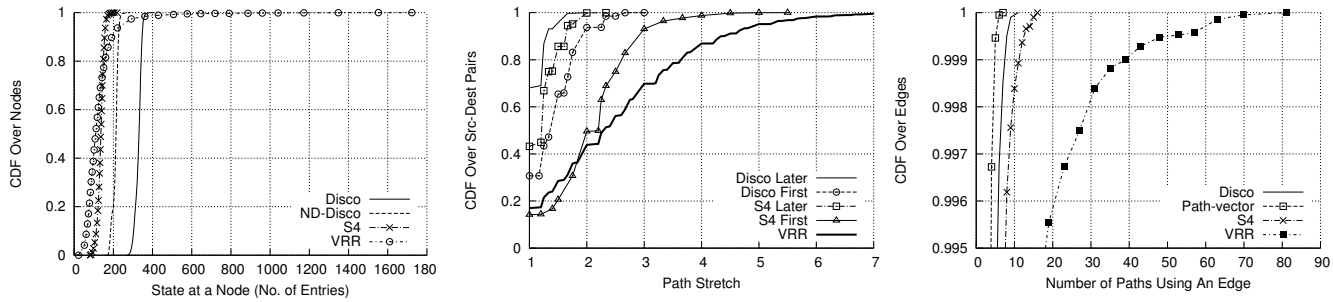



**Figure 4: State (left), stretch (middle) and congestion (right) comparisons between Disco, VRR and S4 over a 1,024-node $G(n, m)$ random graph.**

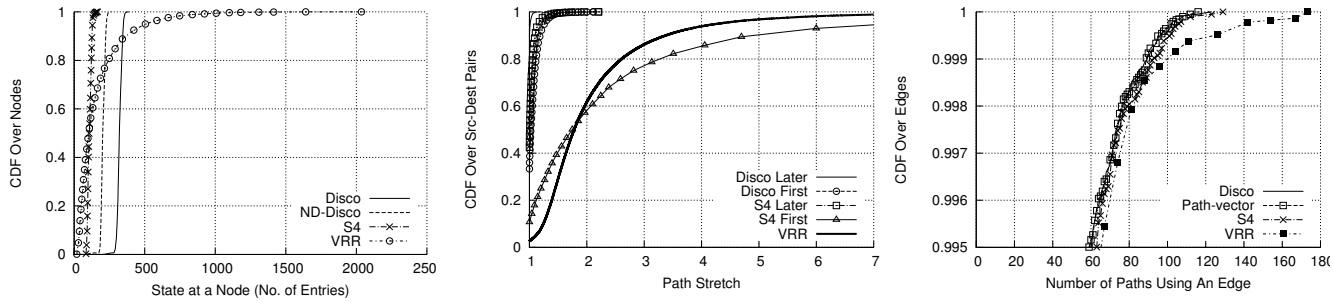



**Figure 5: State (left), stretch (middle) and congestion (right) comparisons between Disco, VRR and S4 over a 1,024-node Geometric random graph.**

| | AS-Level | Router-level | Geometric-16384 | GNM-16384 |
|---|---|---|---|---|
| No Shortcutting | 1.403 | 1.301 | 1.051 | 1.351 |
| To-Destination Shortcuts | 1.271 | 1.191 | 1.005 | 1.285 |
| Shorter{ReversePath, ForwardPath} | 1.327 | 1.229 | 1.026 | 1.266 |
| No Path Knowledge | 1.153 | 1.092 | 1.002 | 1.179 |
| Up-Down Stream | 1.022 | 1.041 | 1.004 | 1.263 |
| Using Path Knowledge | 1.007 | 1.015 | 1.002 | 1.159 |

**Figure 6: Effect of shortcutting strategies: Mean stretch for different shortcutting heuristics.**

**Figure 7: Comparison of state at a node for the Router-level Internet topology. S4 does better on average, but severely breaks worst-case bounds.**

| | No. of Entries | | Bytes (IPv4) | | Bytes (IPv6) | |
|---|---|---|---|---|---|---|
| | Mean | Max | Mean | Max | Mean | Max |
| S4 | 3123.9 | 40339.0 | 18.31 | 236.36 | 54.93 | 709.084 |
| ND-Disco | 3619.88 | 4310.0 | 21.15 | 34.06 | 63.43 | 100.10 |
| Disco | 6592.42 | 7309.0 | 53.03 | 61.608 | 165.31 | 188.20 |

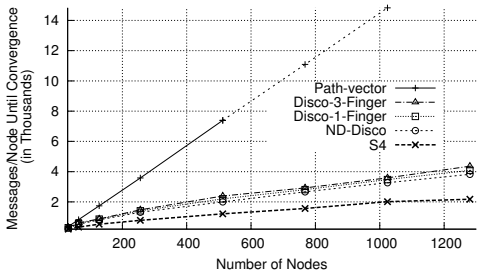



**Figure 8: Mean messages per node sent until convergence in path vector, S4, NDDisco and Disco (with 1 and 3 fingers for address dissemination) for $G(n, m)$ graphs of increasing size. The curve for path vector has been linearly extrapolated beyond 512 nodes.**

is likely that this would qualitatively change the protocols' stretch.

After the first packet, Disco does slightly better than S4 on random and geometric graphs, both perform similarly on the router-level topology, and S4 does better on the AS-level graph.

Fig. 4 (middle) and Fig. 5 (middle) compare stretch in Disco, S4 and VRR over a $G(n, m)$ random graph and a geometric random graph with latency values. Like state, VRR provides no bounds on stretch. The maximum stretch values seen for the first packets in the geometric random graph are 2.4 for Disco, 30 for S4, and 39 for VRR.

As described in §4.2, the above results use the "No Path Knowledge" shortcutting heuristic. Fig. 6 shows the relative effect of different shortcutting heuristics. Using Path Knowledge, the stretch can be brought very close to 1 (last row of table).

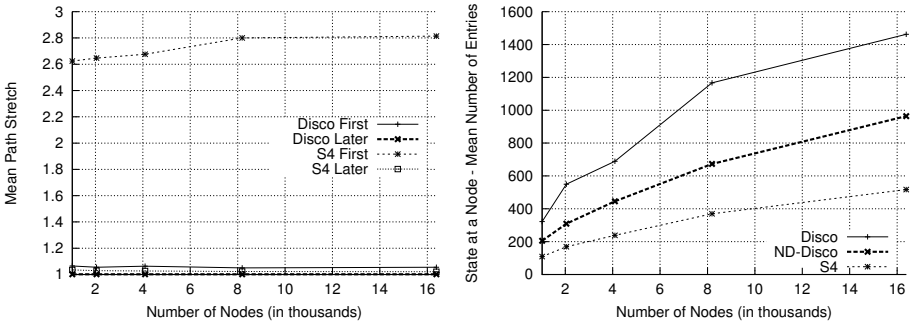
**Control messaging.** We compare messaging costs during initial convergence only, leaving continuous churn to future work. The results are shown in Fig. 8. ND-Disco's messaging overhead is slightly greater than S4's, reflecting its somewhat larger vicinities as discussed above. However, Disco has only a small amount of additional messaging to support routing on flat names with low stretch, demonstrating the efficiency of our dissemination protocol. We show results for 1 and 3 outgoing fingers per node in the address dissemination overlay network. The use of a larger number of fingers leads to lower overlay diameter, and hence faster convergence, at the cost of increased messaging. For instance, for a 1024-node $G(n, m)$ topology, with each node picking 1 outgoing finger, the average and maximum distances traveled by address announcements were measured to be 5.77 and 24 respectively, while picking 3 random fingers reduced these numbers to 3.04 and 16. At the same time, the number of messages increased by 3.3%.

**Congestion.** To compute congestion, we have each node route to a random destination and count the number of times each edge is used. One might have guessed that the compact routing schemes would have high congestion since many routes pass near landmarks. However, Fig. 4 (right) and Fig. 5 (right) show that in synthetic topologies, congestion is surprisingly close to that of shortest-path routing. VRR shows higher congestion than all the three other schemes. On the AS-level Internet topology (Fig. 10), Disco does experience more congestion for a few edges than shortest-path routing.

**Scaling.** Fig. 9 shows how Disco, NDDisco and S4 scale with increasing number of nodes $n$ in geometric random graphs, showing mean stretch and mean state. S4's first-packet stretch remains high, but for the rest of the curves, the stretch is similarly low and close to 1. Routing state grows as $\tilde{O}(\sqrt{n})$.

**Accuracy of static simulation.** We used a static simulation of the network's state after convergence to calculate state, stretch and congestion results for large topologies. Our comparison of results from both the static simulator and the full discrete event simulator shows that the static simulator achieves good accuracy. For instance, for the 1024-node random graph, the difference between mean stretch as measured by the static simulator is within 0.9% for Disco's later packets and

**Figure 9: Disco vs. S4: mean stretch (left) and mean state (right) in geometric random graphs of increasing size.**



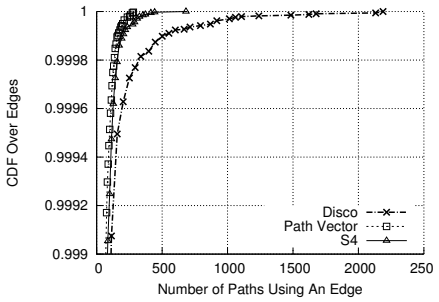




**Figure 10: On the AS-level Internet topology, a small fraction (0.05%) of edges face significantly more congestion than shortest-path routing.**

0.7% for S4's later packets. Both stretches are inflated by similarly small amounts.

**Error in Estimating Number of Nodes.** The previous results assume all nodes know the value of $n$. Here, we inject random errors of up to 60% in this estimation. With 60% random error, across 5 runs on the 1024-node random graph, only one node failed to find in its vicinity a node in only one of the sloppy groups, and hence failed to reach all destinations in that group. With 40% random error, all nodes were able to reach all nodes and mean stretch increased marginally by 0.6% from 1.253 to 1.261. Note that this is an extreme case since we can ensure error is much lower than 40%; §4.1.

## 6. CONCLUSION

Traditionally, hierarchy has been the only way to scale general-purpose routing. Hierarchy has led to inefficiency in routes, and the use of location-dependent addresses which complicate mobility and management. This paper stands in a long line of work which has progressively brought compact routing, an algorithmically promising approach which eschews hierarchy, closer to practical reality. Disco takes another step forward by providing distributed and dynamic routing on flat names with guaranteed scalability and low stretch.

One area of future work deals with improving our stretch bounds and offering different tradeoffs. In particular, is it possible to reduce the worst-case first-packet stretch from 7 to the optimal 3 in a distributed way? Disco has chosen one point in the state/stretch tradeoff space, with $\tilde{O}(\sqrt{n})$ state and stretch $\leq 3$ for packets after the first; can we translate other tradeoff points to a distributed setting for name-independent routing?

Another significant outstanding question is to what extent Disco can support policy routing in the Internet. In many ways, Disco can provide a significant amount of flexibility. For example, although Disco chooses landmarks randomly, its state and stretch stretch bounds require only that each node has at least one landmark within its vicinity and that there are $\tilde{O}(\sqrt{n})$ total landmarks. These rules would permit an operator to choose landmarks in non-random ways, for example to pick a more well-provisioned landmark, ensure that a node's landmark is within its own domain, or use a landmark service supplied by a network provider. And to maintain a globally scalable infrastructure with $\tilde{O}(\sqrt{n})$ total landmarks, landmark identifiers could be purchased from or allocated by a registry, much as AS numbers are today. However, policies also pose challenges. Disco assumes that the route $v \rightsquigarrow \ell_v$ can also be used in the reverse direction in $v$'s address in order to route $\ell_v \rightsquigarrow v$, and assumes similar reversibility for vicinity routes (when checking the destination's vicinity for a path from the source), which would limit the possible policies. A second problem is that policy routing can significantly lengthen paths; routing through the landmark nearest to a destination many not provide a stretch guarantee for general policies, even when the route length is compared to the shortest policy-compliant path. Resolving these challenges would be an interesting area of future work.

We thank our shepherd, Bruce Maggs, and the anonymous reviewers for helpful comments. This work was supported by NSF CNS 10-17069. The second author was a visiting researcher at Intel Labs Berkeley during part of this project.

# 7. REFERENCES


[1] I. Abraham, A. Badola, D. Bickson, D. Malkhi, S. Maloo, and S. Ron. Practical locality-awareness for large scale information sharing. In *Proc. IPTPS*, 2005.

[2] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. *Lecture notes in computer science*, pages 305–319, 2004.

[3] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 20–24, New York, NY, USA, 2004. ACM.

[4] I. Abraham, D. Malkhi, and O. Dobzinski. Land: Stretch (1 + e) locality-aware networks for dhts. In *Proc. SODA*, pages 550–559, 2004.

[5] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *ACM SIGCOMM*, Seattle, WA, August 2008.

[6] M. Arias, L. J. Cowen, K. A. Laing, R. Rajaraman, and O. Taka. Compact routing with name independence. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 184–192, New York, NY, USA, 2003. ACM.

[7] B. Awerbuch, A. Bar-noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *Proc. STOC*, pages 479–489. ACM Press, 1989.

[8] J. Brutlag. Speed matters for Google web search, June 2009. `http://code.google.com/speed/files/delayexp.pdf`.

[9] M. Caesar, M. Castro, E. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTs. *ACM SIGCOMM Computer Communication Review*, 36(4):362, 2006.

[10] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: routing on flat labels. In *ACM SIGCOMM*, 2006.

[11] D. Clark. The design philosophy of the DARPA Internet protocols. In *Proc. SIGCOMM*, page 114, 1988.

[12] D. Clark, R. Braden, A. Falk, and V. Pingali. Fara: reorganizing the addressing architecture. *SIGCOMM Comput. Commun. Rev.*, 33(4):313–321, 2003.

[13] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID separation protocol (LISP). In *Internet-Draft*, March 2009.

[14] R. Fonseca, S. Ratnasamy, J. Zhao, C. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *NSDI*, May 2005.

[15] B. A. Ford. *UIA: A Global Connectivity Architecture for Mobile Personal Devices*. PhD thesis, Massachusetts Institute of Technology, September 2008.

[16] P. Fraigniaud and C. Gavoille. Memory requirement for universal routing schemes. In *PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pages 223–230, New York, NY, USA, 1995. ACM.

[17] C. Gavoille. An overview on compact routing schemes. In *Presentation at 2nd Research Workshop on Flexible Network Design*, October 2006. `http://www.aladdin.cs.cmu.edu/workshops/netdes2/slides/gavoille.pdf`.

[18] C. Gavoille and M. Gengler. Space-efficiency for routing schemes of stretch factor three. *J. Parallel Distrib. Comput.*, 61(5):679–687, 2001.

[19] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *ACM SIGCOMM*, 2009.

[20] M. Gritter and D. R. Cheriton. An architecture for content routing support in the internet. In *USENIX Symposium on Internet Technologies and Systems*, March 2001.

[21] P. Jokela, P. Nikander, J. Melen, J. Ylitalo, and J. Wall. Host identity protocol-extended abstract. In *Wireless World Research Forum*, February 2004.

[22] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proc. STOC*, pages 654–663, New York, NY, USA, 1997. ACM.

[23] B. Karp and H. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proc. MobiCom*, pages 243–254. ACM, August 2000.

[24] E. Karpilovsky and J. Rexford. Using forgetful routing to control BGP table size. In *CoNEXT*, 2006.

[25] R. Kessler and J. Schwarzmeier. CRAY T3D: A new dimension for Cray Research. *Compcon Spring'93, Digest of Papers.*, pages 176–182, 1993.

[26] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *ACM SIGCOMM*, Seattle, WA, August 2008.

[27] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1(3):155–174, January 1977.

[28] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM Computer Communication Review*, 37(4):192, 2007.

[29] D. Krioukov, K. Fall, and X. Yang. Compact routing on Internet-like graphs. In *IEEE INFOCOM*, volume 1, pages 209–219. Citeseer, 2004.

[30] D. Krioukov, kc claffy, K. Fall, and A. Brady. On compact routing for the internet. *ACM SIGCOMM Computer Communication Review*, 37(3):52, 2007.

[31] K. Levchenko, G. Voelker, R. Paturi, and S. Savage. XL: An efficient network routing algorithm. In *ACM SIGCOMM*, pages 15–26, August 2008.

[32] G. S. Manku, M. Bawa, P. Raghavan, and V. Inc. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, pages 127–140, 2003.

[33] G. S. Manku, M. Naor, and U. Wieder. Know thy neighbor's neighbor: the power of lookahead in randomized p2p networks. In *Proc. STOC*, pages 54–63, 2004.

[34] Y. Mao, F. Wang, L. Qiu, S. Lam, and J. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *Proc. NSDI*, April 2007.

[35] D. Mazieres, M. Kaminsky, M. Kaashoek, and E. Witchel. Separating key management from file system security. In *ACM SOSP*, December 1999.

[36] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys*, 2004.

[37] D. Peleg and E. Upfal. A tradeoff between space and efficiency for routing tables. In *Proc. STOC*, pages 43–52, New York, NY, USA, 1988. ACM.

[38] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. SPAA*, pages 311–320, New York, NY, USA, 1997. ACM.

[39] A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proc. Middleware*, 2001.

[40] M. Särelä, T. R. Aho, and T. Tarkoma. RTFM: Publish/subscribe internetworking architecture. *ICT Mobile Summit*, 2008.

[41] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names, July 2010. `http://www.cs.illinois.edu/homes/singla2/compactrouting.pdf`.

[42] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proc. SIGCOMM*, pages 73–86. ACM New York, NY, USA, 2002.

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

[44] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. SPAA*, pages 1–10. ACM, 2001.

[45] P. F. Tsuchiya. The landmark hierarchy: a new hierarchy for routing in very large networks. In *Proc. SIGCOMM*, pages 35–42, New York, NY, USA, 1988. ACM.

[46] M. Walfish, H. Balakrishnan, and S. Shenker. Untangling the Web from DNS. In *Proc. NSDI*, page 17. USENIX Association, 2004.

[47] C. Westphal and J. Kempf. A compact routing architecture for mobility. In *Proc. MobiArch*, pages 1–6. ACM, 2008.

[48] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Colleen Shannon, Matthew Luckie, and kc claffy. The CAIDA IPv4 Routed /24 Topology Dataset. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml` (accessed on 2009-11).

[49] Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, kc claffy, and Colleen Shannon. The IPv4 Routed /24 AS Links Dataset. `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml` (accessed on 2009-11).

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