VICINITY: A Pinch of Randomness Brings out the Structure

Spyros Voulgaris[1] and Maarten van Steen[1]

VU University, Amsterdam, The Netherlands
{spyros,steen}@cs.vu.nl

Abstract. Overlay networks are central to the operation of large-scale decentralized applications, be it Internet-scale P2P systems deployed in the wild or cloud applications running in a controlled—albeit large-scale—environment. A number of custom solutions exist for individual applications, each employing a tailor-made mechanism to build and maintain its specific structure. This paper addresses the role of randomness in developing and maintaining such structures. Taking VICINITY, a generic overlay management framework based on self-organization, we explore tradeoffs between deterministic and probabilistic decision-making for structuring overlays. We come to the conclusion that a pinch of randomness may even be needed in overlay construction, but also that much randomness or randomness alone is not good either.

1 Introduction

Does randomness matter? In this paper we claim it does, and, in fact, that incorporating randomness into distributed algorithms may even be necessary. We do not claim that randomness is necessary for all algorithms (which would clearly be wrong), but that for many large-scale distributed algorithms it is important to strive for simplicity through loose control. What is lost is determinism and the potential to formally prove correctness. Instead, at best only statistical properties can be shown to hold, but what can be achieved is that those properties emerge from very simple principles. A fundamental principle being that decisions concerning selection, of whatever kind, are sometimes random.

To substantiate our claim, we consider the influence of randomness in distributed gossiping algorithms. Gossiping is a well-known, and simple technique, widely deployed for a range of applications, including data replication, information dissemination, and system management. Gossiping is often deterministic: the rules for selecting whom to gossip with and what to gossip are strict, with no probabilistic element. On the other hand, there are also many gossiping algorithms that incorporate probabilistic decision-making, yet lack an examination of *why* such decision-making is so effective.

We have no general answer to where the effectiveness of randomness comes from, yet we believe such understanding is crucial for designing large-scale distributed systems. As a step toward such understanding, we concentrate in this paper on deploying a gossiping algorithm called VICINITY, for constructing overlay networks. It is not our purpose to advocate our solution to overlay construction. Instead, we use VICINITY as a

framework to demonstrate how crucial incorporating randomness is. More specifically, we show that there is a subtle balance to be sought between deterministic and probabilistic decision-making. A pinch of randomness is enough, too much randomness will spoil matters.

Our main contribution is systematically exploring the effect of randomness in gossip-based overlay construction. This brings us to the conclusion that such exploration can be crucial and that deciding in advance on the amount of randomness is diffcult, if not impossible. As a side-effect of this exploration, we present VICINITY, a novel gossiping algorithm that can be deployed for a wide range of applications.

The rest of the paper is organized as follows. Section 2 defines our system model. Section 3 presents the VICINITY protocol, starting from its intuition, a baseline model, and the detailed design decisions that lead to the complete version of the protocol. Section 4 sheds some light on the individual roles of determinism and randomness. Section 5 offers an evaluation of VICINITY through two scenarios that portray the interplay between determinism and randomness and highlight their individual strengths and weaknesses. Section 6 discusses related work, and Section 7 communicates our overall conclusions from this work.

2 System Model

The Network We consider a set of N nodes connected over a routed network infrastructure. Each node has a *profile*, containing some application-specific data of the node, determining the node's neighbors in the target structure. Such a profile could contain a node's geographic coordinates, a vector of preferences, social network information, or in general any other metric that the application uses for defining the target structure.

Knowledge regarding neighbors is stored and exchanged by means of node *descriptors*. The descriptor of a given node can be generated exclusively by that very node, but it can be freely handed by any third node to any other. The descriptor of a node is a tuple containing the following three fields:

1. the node's **address** (i.e., IP address and port)
2. the descriptor's **age** (a numeric field)
3. the node's application-specific **profile**

We consider that nodes are connected over a network that supports routing. That is, *any* node can send a message to *any* other, provided that the sender knows the receiver's *address* (i.e., IP address and port, on the Internet).

To enable communication with other nodes, each node maintains a small dynamic list of neighbors, called its *view*, V. A node view is essentially a list of descriptors of the node's neighbors. Node views have a small fixed length, ℓ. Their contents are dynamic, and are updated in an epidemic fashion through pairwise node communication. Although this is not binding, for simplicity we will consider that all nodes have the same view length.

The network is inherently dynamic and unreliable. Nodes can join, leave, or crash at any time and without prior notice. In particular, we make no distinction between node crashes and node leaves. Additionally, nodes are also free to dynamically update their

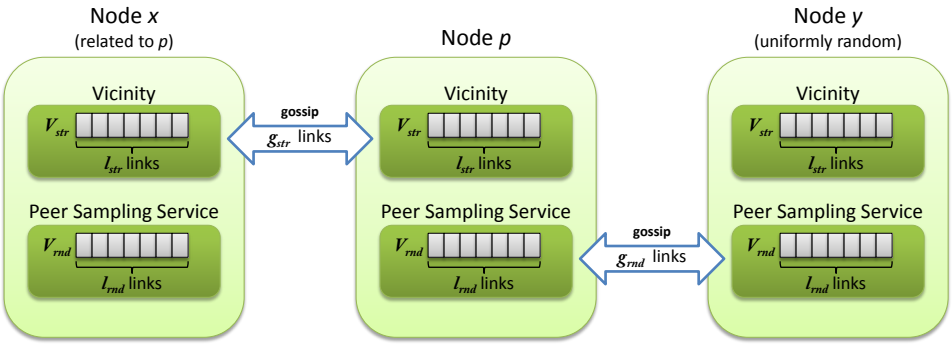


Fig. 1. The VICINITY framework.

profiles. Messages may be lost, or delayed. Byzantine behavior is beyond the scope of this work.

Finally, we consider that nodes participate in a *peer sampling service* [4], which provides them with a continuous stream of links to nodes picked uniformly at random among all *alive* nodes. Peer sampling protocols form a fundamental ingredient of many peer-to-peer applications nowadays, they are completely decentralized, and they have shown to be remarkably inexpensive.

As VICINITY strives for creating *structure*, we will be referring to its view as V_{str}, to its view's length as ℓ_{str}, and to its gossip length (i.e., the number of descriptors exchanged in each direction in a gossip interaction) as g_{str}. Likewise, as the peer sampling service is responsible for *randomness*, its view, view length, and gossip length will be referred to as V_{rnd}, ℓ_{rnd}, and g_{rnd}, respectively.

The Target Overlay We also consider a *selection function* $\text{SELECT}(p, \mathcal{D}, k)$, that, given the descriptor of node p and a set $\mathcal{D}$ of node descriptors, returns the set of k descriptors (or all of them, if $|\mathcal{D}| < k$) that best approximate p's outgoing links in the target structure. The selection is based on node profiles. We assume function SELECT to be globally known by all nodes in the system.

The selection function essentially defines the target structure. Each node p aims at eventually establishing links to the "best" ℓ_{str} nodes, as defined by the outcome of $\text{SELECT}(p, \mathcal{D}_p^*, \ell_{str})$, where $\mathcal{D}_p^*$ is the set of descriptors of all nodes in the network excluding p.

Often, the selection function SELECT is based on a globally defined node proximity metric. That is, $\text{SELECT}(p, \mathcal{D}, k)$ sorts all descriptors in $\mathcal{D}$ with respect to their proximity to node p, and selects the k closest ones. Typical proximity metrics include semantic similarity, ID-based sorting, domain name proximity, geographic- or latency-based proximity, etc. Some applications may apply composite proximity metrics, combining two or more of the above. In certain cases, though, selecting appropriate neighbors involves more than a mere sorting based on some metric, typically when a node's sig-

nificance as a neighbor depends not only on the its proximity to a given node, but also on which *other* nodes are being selected.

We assume that the selection function exhibits some sort of *transitivity*, in the sense that if node b is a "good" selection for node a ($a \overset{\text{SELECT}}{\dashrightarrow} b$), and c is a "good" selection for b ($b \overset{\text{SELECT}}{\dashrightarrow} c$), then c *tends* to be a "good" selection for a too ($a \overset{\text{SELECT}}{\dashrightarrow} c$). Generally, the "better" a selection node q is for node p, the more likely it is that q's "good" selections are also "good" for p.

This transitivity is essentially a correlation property between nodes sharing common neighbors, embodying the principle "my friend's friend is also my friend". Surely, this correlation is fuzzy and generally hard to quantify. It is more of a desired property rather than a hard requirement for our topology construction framework. The framework excels for networks exhibiting strong transitivity. However, its efficiency degrades as the transitivity becomes weaker. In the extreme case that no correlation holds between nodes with common neighbors, related nodes eventually discover each other through random encounters, although this may take a long time.

3 The VICINITY Protocol

3.1 VICINITY: The Intuition

The goal is to organize all VICINITY views so as to approximate the target structure as closely as possible. To this end, nodes regularly exchange node descriptors to gradually evolve their views towards the target. When gossiping, nodes send each other a subset of their views, of fixed small length g_{str}, known as the *gossip length*. The gossip length is the same for all nodes.

From our previous discussion, we are seeking a means to construct, for each node and with respect to the given selection function, the optimal view from all nodes currently in the system. There are two sides to this construction.

First, based on the assumption of transitivity in the selection function, SELECT, a node should explore the nearby nodes that its neighbors have found. In other words, if b is in a's VICINITY view, and c is in b's view, it makes sense to check whether c would also be suitable as a neighbor of a. Exploiting the transitivity in SELECT should then quickly lead to high-quality views. The way a node tries to improve its VICINITY view resembles *hill-climbing* algorithms [9]. However, instead of trying to locate a single optimal node, here the objective is to optimize the selection of a whole set of nodes, namely the view. In that respect, VICINITY can be thought of as a distributed, collaborative hill-climbing algorithm.

Second, it is important that *all* nodes be examined. The problem with following transitivity alone is that a node will be eventually searching only in a single cluster of related nodes, possibly missing out on other clusters of also related—but still unknown—peers, in a way similar to getting locked in a local maximum in hill-climbing algorithms. Analogously to the special "long" links in small-world networks [12], a node needs to establish links outside its neighborhood's cluster. Likewise, when new nodes join the network, they should easily find an appropriate cluster to join. These issues call for a randomization of candidates for including in a view.

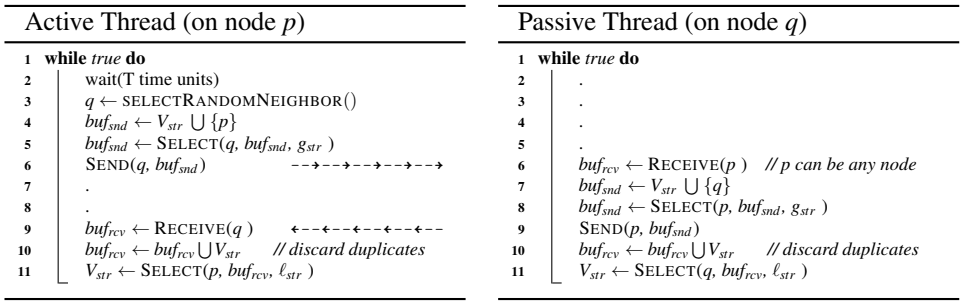


Fig. 2. Baseline version of the VICINITY protocol.

In our design we decouple these two aspects by adopting a two-layered gossiping framework, as can be seen in Figure 1. The lower layer is the peer sampling service, responsible for maintaining a connected overlay and for periodically feeding the top-layer protocol with nodes uniformly randomly selected from the whole network. In its turn, the top-layer protocol, called VICINITY, is in charge of discovering nodes that are favored by the selection function. Each layer maintains its own, separate view, and communicates to the respective layer of other nodes.

3.2 VICINITY: Baseline Version

To better grasp the principal operation of the protocol, we first present a baseline version of VICINITY, shown in Figure 2. In this baseline version, each node periodically contacts a random node from its view, and the two nodes send each other the best—with respect to the receiver's profile—g_{str} neighbors they have in their views. Note that this baseline version of VICINITY is completely equivalent to the related T-MAN protocol [5].

As can be seen in the pseudocode of Figure 2, each node, p, periodically picks from its view a random node, q, to gossip with (line 3). It then applies the SELECT function to select the g_{str} nodes that are best for q, from the union of its own view and p itself (lines 4-5), and sends them to q (line 6). Upon reception of p's message, q selects the g_{str} best nodes for p among all nodes in its view and q itself (lines 7-8), and sends them back to p (line 9). Finally, each node updates its own view, by selecting the ℓ_{str} best neighbors out of its previous view and all received descriptors (line 11).

Note that the code for selecting and sending descriptors to the other side is symmetric for the two nodes (lines 4-6 vs. lines 7-9), as well as the code for merging the received descriptors to the current view (lines 11).

Each node essentially runs two threads. An *active* one, which periodically wakes up and initiates communication to another node, and a *passive* thread, which responds to the communication initiated by another node.

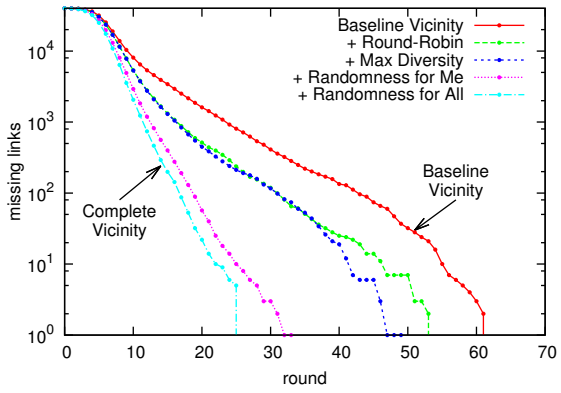


Fig. 3. Self-organization in a 100×100 torus, demonstrating the performance for different versions of VICINITY, ranging from the baseline to the complete one.

3.3 VICINITY: Fine-tuning the nuts and bolts

A number of interesting design choices can substantially boost the performance of the baseline VICINITY protocol. In this section, we will motivate them and demonstrate them in parallel. For our demonstration we will consider a sample testbed, simulated on PeerNet [6], an open-source simulation and emulation framework for peer-to-peer networks written in Java, branching the popular PeerSim simulator [7].

Our testbed consists of a network of 10,000 nodes, assigned distinct 2D coordinates from a 100×100 grid, and whose aim is to self-organize in the respective torus overlay, starting from an arbitrary random topology. Nodes maintain a short view of ℓ_{str}=12 descriptors each, which is initially filled with 12 neighbors picked uniformly at random from the whole network. When gossiping, nodes send g_{str}=12 descriptors to each other. The selection function selects, out of a given set, the k neighbors that are the closest to the reference node in Euclidean space. The goal of a node is to discover its four closest nodes out of the whole network, that is, to get their descriptors in its view. For example, the node with coordinates $(20,40)$ should get nodes $(19,40)$, $(21,40)$, $(20,39)$, and $(20,41)$ among its neighbors. We consider space to wrap around the edges of the grid, resulting in a torus topology. For example, the four closest nodes for node $(0,0)$ are $(99,0)$, $(1,0)$, $(0,99)$, and $(0,1)$.

Figure 3 plots the number of target links that are missing from all nodes' views, collectively. Initially, this accounts to 40,000 links, i.e., four for each of the 10,000 nodes. The red plot corresponds to the baseline version of VICINITY, detailed in the previous section. Clearly, target links are being discovered at exponential speed, and within 61 rounds nodes have self-organized to a complete torus structure. Nevertheless, as we see, the baseline is the slowest of all five versions shown.

Round-robin neighbor selection The first improvement concerns the policy for selecting which neighbor to gossip with. Rather than picking from one's view at random,

we impose a *round robin* selection of gossip partners. The motivation behind this policy is twofold.

First, contacting one's neighbors in a round-robin order improves the node's chances to optimize its view faster, by increasing the number of *different* potentially good neighbors the node encounters. It is not hard to envisage that probing a single neighbor multiple times in a short time frame has little value, as the neighbor is unlikely to have new useful information to trade every time. In contrast, maximizing the intervals at which a given neighbor is probed, maximizes the potential utility of each gossip exchange. Given the rather static nature of a node's VICINITY view when converged, this is achieved by visiting neighbors in a round-robin fashion.

The second motivation for the round-robin policy is that, in the case of a dynamic network, it serves garbage collection of obsolete node descriptors. A descriptor may become obsolete as a result of network dynamics, if the node it points at is no longer alive. By picking neighbors in round-robin order, neighbors are being contacted in roughly uniform time periods, preventing any single—and possibly obsolete—descriptor from lingering indefinitely in a node's view.

The green plot of Figure 3 shows the evolution of the same experiment, with round-robin neighbor selection enabled. The improved performance over the baseline version is evident already from the early rounds of the experiment.

Maximize descriptor diversity Another way to squeeze more benefit out of a single gossip exchange, is to increase the diversity of descriptors exchanged between the nodes. When responding to a node's gossip request, there is no value in sending back descriptors that were also included in that node's message. That node has these descriptors already. This can be very common especially when the network is in a converged or nearly converged state, in which case nodes are highly clustered. In that respect, a node's passive thread should *exclude* all received descriptors from the set of potential descriptors to send back.

The dark blue plot of Figure 3 presents the evolution of the experiment, this time applying both the round-robin and the diversity maximization policies. The plot confirms our reasoning, and shows that the discovery of target links is indeed accelerated, particularly at the stages closer to convergence, as anticipated.

Randomness for me Let us now take a ground-breaking twist in our design. All configurations considered so far have been too narrowly structure-oriented. They all exploit a single input channel of information for improving structure, and that channel is nothing more than other nodes' structure information. We have created a feedback loop on structure for structure! Or rather, a vicious cycle around structure.

Depending on the scenario, this can be a strength or a weakness. Once connected to some "good" neighbors, the chances to be introduced to additional "good" nodes increases. Once, however, connected exclusively to largely irrelevant nodes, navigating towards one's "neighborhood" can be slow, or in certain occasions impossible. We defer this discussion to Section 4.

With the intent of breaking the closed loop on structural information, we introduce *randomness* as a second input channel. Rather than having nodes discover new neigh-

bors exclusively through their current neighbors' structural links, we also offer them the chance to sample nodes from the whole network *at random*.

To this end, we employ CYCLON [10] as a peer sampling protocol, to provide nodes with a stream of random neighbors. In each round, each node's active thread pulls the random descriptors provided by its CYCLON instance, merges them with it normal VICINITY view, and filters the union through the SELECT function to keep the best ℓ_{str} neighbors. This way, if CYCLON encounters a good neighbor by chance, that neighbor is pick up by VICINITY to improve its view.

For the sake of a fair comparison, we maintain the number of descriptors exchanged per round the same as in the baseline configuration, that is, 12 descriptors per round. However, now we exchange g_{str}=6 descriptors on behalf of VICINITY, and another six descriptors on behalf of CYCLON. This creates precisely the same bandwidth requirements as in the previous configurations, although distributed in a double number of half-sized packets.

The magenta plot of Figure 3 confirms that the configuration combining structure with randomness significantly outperforms all previous versions. It is worth emphasizing that the *rate* of discovering target links is significantly faster for the whole extent of the experiment, from its early stages until full convergence, despite the fact that only six links are exchanged per round by VICINITY as opposed to 12 in previous configurations.

Randomness for all A final optization is to borrow the random links obtained through CYCLON not only for improve a node's own structure links, but also to improve the quality of links it sends to other nodes.

The dark blue plot of Figure 3 clearly shows that this optimization further improves performance. This last configuration constitutes the complete VICINITY protocol, and will be the one used by default for the rest of the paper, unless otherwise mentioned.

3.4 VICINITY: The Complete Protocol

The complete VICINITY protocol is presented—in pseudocode—in Figure 4. The rest of this section discusses the differences to the baseline protocol.

The round-robin neighbor selection policy is implemented by means of the *age* field in descriptors. The age of a descriptor gives an *approximate* estimation of how many rounds ago that descriptor either (i) was introduced in that node's VICINITY view, or (ii) was last used by the node for gossiping with the respective neighbor. Neighbors of higher age are given priority when choosing a neighbor for gossiping (line 3), and subsequently their age is zeroed, which results in a round-robin selection policy.

To approximate the number of rounds some descriptor has been in a node's view, any new descriptor entering a view is initialized with zero age (lines 9–active, 13–both), and the ages of all descriptors in the view are incremented by one once per round (line 5). Also, when a node is selected for gossiping (line 3), it is also *removed* from the view (line 4), as a means for garbage collection of descriptors. If that neighbor is still alive and responds, its fresh descriptor (with age zero) will be inserted anew to the view.

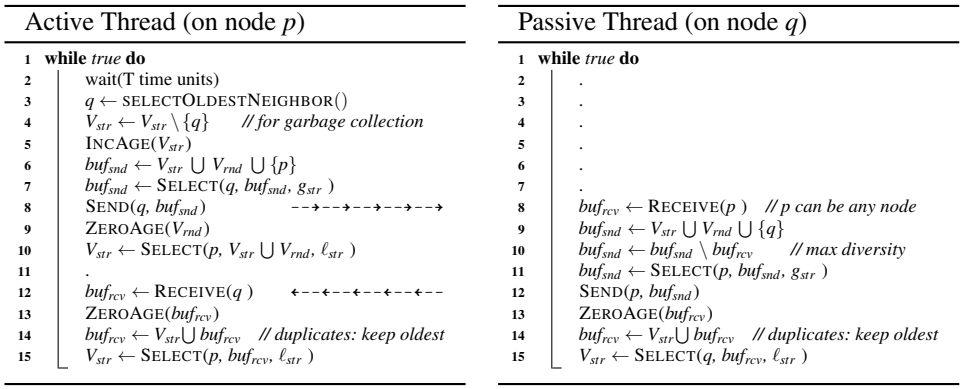


Fig. 4. The complete VICINITY protocol.

The role of randomness can be seen in lines 6–active and 9–passive, where random neighbors are also considered in the message to send to the other peer, as well as in line 10–active, where a node pulls "good" neighbors from its randomized view into its structured view.

From this point on, by VICINITY we will be referring to the complete version of the protocol, including all the design optimizations presented so far.

4 How much Randomness is Enough?

Randomness is good. At least for the specific scenario of the previous section. But how general can this claim be? How good is randomness in other scenarios? Just good, or rather necessary? How much randomness is "enough", and how much can it assist in structuring? Although it is infeasible to give a universal rule to quantitatively assess the value of randomness, in this section we aim at shedding some light at these questions.

To answer these questions, we delve into the principles governing self-organization, and we distinguish the specific roles of determinism and randomness in it.

4.1 The Role of Determinism

To explore the role of determinism, alone, isolated from the effects of randomness, let us consider self-organization without randomness, relying exclusively on structure. To further isolate our reasoning from the effects of randomness, including pseudo-randomness due to nodes continuously replacing their links *during* the process of convergence, it may help to think of fresh nodes joining an *already converged* network.

The whole operation of self-organization relies on the ability to periodically compare potential neighbors, and on being able to determine which ones are a step closer to your targets. We are looking, therefore, at some form of routing or orientation property in the target overlay.

For simplicity, let us consider a very trivial case. The whole network has converged, except for a single node, x. Node x has one target, z, and currently has exactly one

neighbor, y. Imagine, for instance, a fresh node x joining an already converged network using an arbitrary node y as its bootstrap node. For self-organization to be successful, x should be able to reach z through y, y's neighbors, y's neighbors' neighbors, and so on. And this should be the case for *any* y and *any* z. This dictates the first required property for self-organization based exclusively on structure to be correct: *the target topology should form a strongly-connected graph.*

This, however, is not sufficient. Even if a directed path from y to z exists, say consisting of nodes y_1, y_2, ..., y_k, the selection function should be such that a call to SELECT$(x, Neighbors(y), g_{str})$ returns a subset of y's neighbors that contains y_1, then a call to SELECT$(x, Neighbors(y_1), g_{str})$ returns a set of nodes that contains y_2, and so on. We will refer to this property as *navigability*, and we state the second required property for correctness: *the given selection function should render the given target overlay navigable.*

Note that navigability is a property of the *combination* of (i) the target overlay and (ii) the selection function. Clearly, a strongly connected target overlay with a selection function that returns "bad" selections, will not let a network self-organize. The other way around, a selection function that works for some particular overlay will not necessarily be sufficient for any overlay. For instance the proximity-based selection function used in Section 3 is excellent for uniformly populated topologies, but it can get some nodes trapped in "local optima" in the presence of a U-shaped gap, a well known problem of greedy geographic routing protocols [1].

4.2 The Triple Role of Randomness

Having discussed the weaknesses of determinism in self-organization, it is not hard to imagine the benefits offered by randomization.

First, maintaining the whole overlay in a single connected partition is the cornerstone of any large-scale decentralized application. This need is even more pressuring in the case of a custom overlay management protocol, as the target overlay may per se consist of multiple distinct components. Keeping the whole overlay connected in a single component allows the joining of new nodes at arbitrary bootstrap points, and generally allows the reconfiguration of nodes in case of updates to their profiles.

Second, feeding nodes with neighbors picked uniformly at random from the whole overlay, prevents them from getting indefinitely stuck in local optima. Similarly to hill climbing algorithms, random sampling is crucial at helping nodes reach their global optimum.

Finally, even in target overlays and selection functions that guarantee a strongly connected, navigable target overlay, the diameter of that overlay is often large. When new nodes join a converged overlay at an arbitrary bootstrap node, it may take them long to gradually navigate to their optimal neighbors. Having a continuous stream of random samples from the whole network, however, gives them the opportunity to take a shortcut link close to their target, a well-known property of random, complex networks.

5 Evaluation

Given VICINITY's generic applicability, it is practically infeasible to provide an exhaustive evaluation of the framework. Instead, we will focus on the following two test cases that underline its two key components: its reliance on structure and its benefit from randomness:

Two-dimensional Torus This is the same overlay structure we used in Section 3. Nodes are assigned two-dimensional coordinates, and their goal is to establish links to their closest neighbors. Building this target topology is primarily based on the Euclidean proximity heuristic. Informally speaking, the general idea is that nodes gradually improve their views with closer neighbors, which they then probe to find new, even closer ones, eventually reaching their closest neighbors. This emphasizes the utility of the deterministic component of VICINITY.

Clustering Nodes in Groups In this test case, nodes are split up in uncorrelated groups. Each node's goal is to cluster with other nodes of the same group. The key difference with the previous test case is that nodes cannot gradually connect to groups "closer" to their own, as there is no notion of proximity between groups. The target overlay is explicitly clustered in non-connected components, therefore it is neither (strongly) connected nor navigable. Finding a node of the same group can be accomplished only by means of random encounters, which highlights the role of randomness. Once a node of the same group is found, though, the two nodes can grreatly assist each other by sharing their further knowledge of same group neighbors.

5.1 Two-dimensional Torus

Overview We consider a two-dimensional space. We assign each node (x,y) coordinates, such that they are (virtually) aligned in a regular square grid organization. A node's coordinates constitute its profile. Each node's goal is to establish links to its four closest neighbors, to the north, south, east, and west (wrapping around the grid's edges).

The natural choice of a selection function for such a target topology is one that gives preference to neighbors spatially closer to the reference node. More formally, we define the distance between two nodes a and b, with coordinates (x_a, y_a) and (x_b, y_b), respectively, to be their two-dimensional Euclidean distance, assuming that space wraps around the edges to form a torus:

$$dx = min\{|x_a - x_b|,\ \ width - |x_a - x_b|\}$$

$$dy = min\{|y_a - y_b|,\ \ height - |y_a - y_b|\}$$

$$dist(a,b) = \sqrt{dx^2 + dy^2}$$

The selection function $\text{SELECT}(p, \mathcal{D}, k)$ sorts all node descriptors in $\mathcal{D}$ by their distance to the reference node p, and returns the k closest ones.

Figure 5 graphically illustrates the self-organization of a "toy-size" network of 1024 nodes into a torus overlay, depicting snapshots of the overlay at different stages. Nodes'

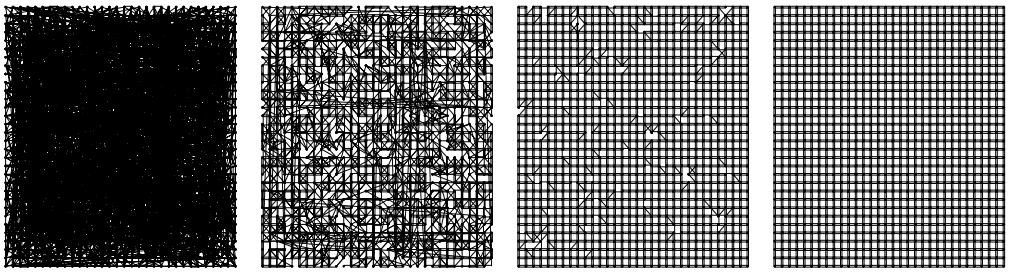


Fig. 5. Self-organization in a 32×32 torus topology.

deterministic and randomized views have been set to a size of six, each. For clarity of the snapshots, only the best four outgoing links of each node are shown in the figure. Note the existence of either one or two lines between two connected nodes. This is because links are directed. A single line denotes a single link only from one node to the other (directionality not shown). A double line means that both nodes have established a link to each other. In the completed target topology (last snapshot) all links are double.

Experimental Analysis Let us now observe the progress of self-organization for different network sizes and protocol configurations. Figure 6 plots the fraction of missing target links as a function of the experiment round, for networks of size 2^{12}, 2^{14}, and 2^{16} nodes, respectively. For each network size we present the progress of five different configurations. For a fair comparison, we have fixed the total number of descriptors exchanged in a single round by a node's active thread to 12.

The thick solid blue and green lines correspond to trading exclusively deterministic or randomized links, respectively. That is, all 12 links exchanged come either from the deterministic view or from the randomized view, respectively. The fine line of a given color corresponds to a very close configuration to its solid line counterpart, where just *one* link has been reserved for trading neighbors of the other view type. E.g., a fine blue line corresponds to the settings g_{str}=11 and g_{rnd}=1. Finally, the red line corresponds to an equally balanced use of determinism and randomness: six links are being traded per round for each view type.

A number of observations can be made from these graphs. Most importantly, we easily identify determinism as the primary component responsible for efficient self-organization. On the contrary, when randomness is used alone (solid green line), it performs several orders of magnitude worse than the other protocol configurations, whose performances are comparable to each other. This indicates that, for the given target topology, the crucial element accelerating self-organization is determinism.

It is not hard to see why using randomness alone is so inefficient. A node's only chance to find a target neighbor is if that neighbor shows up in its peer sampling service view, which is periodically refreshed with random nodes. In other words, a node is

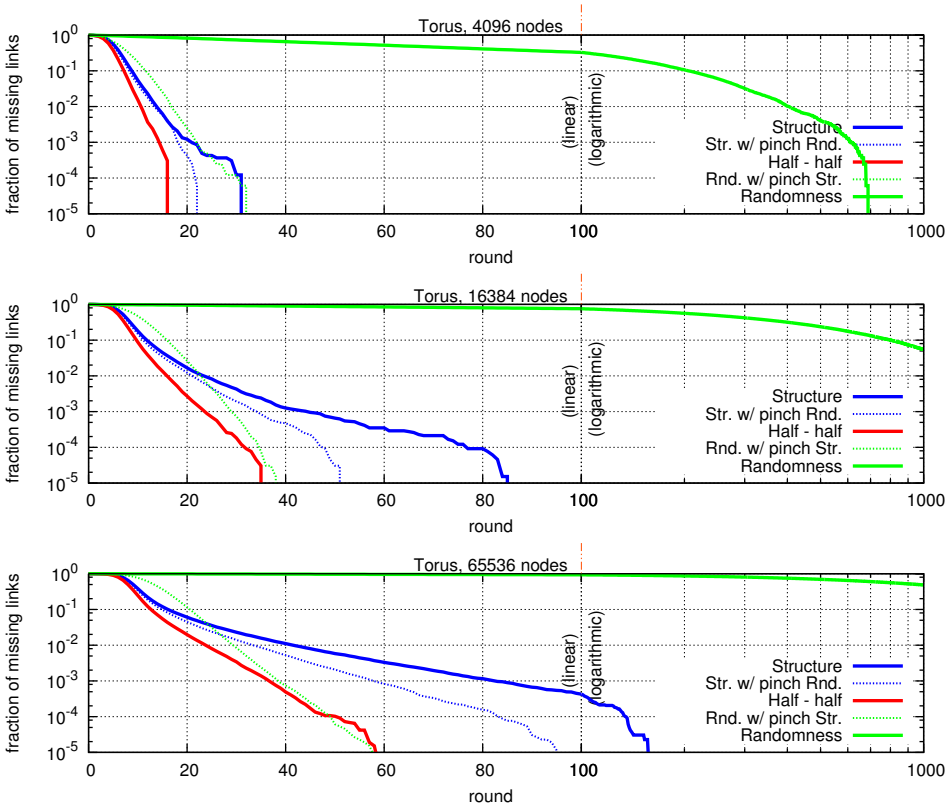
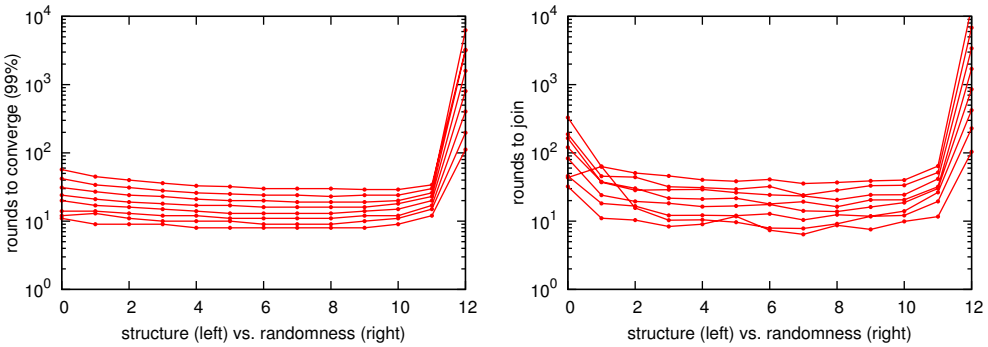


Fig. 6. Progress of self-organization in a torus overlay, for different configurations of VICINITY and a total gossip length (g_{str}+g_{rnd}) fixed to 12.

fishing for target neighbors blindly. As expected, its time to converge increases significantly as the size of the network grows, since the probability of spotting a target link at random diminishes.

Note that just a "pinch" of structure in a nearly random-only configuration (fine green line) brings a dramatic improvement to the outcome. This emphasizes the importance of structure, particularly in an overlay as navigable as a torus topology. In this scenario, a node has plenty of random input, while that single structured link deterministically brings it closer to its target neighborhood in each round.

When determinism is in exclusive control (blue line), convergence comes fast as node views deterministically improve in each round. An important observation, though, is that in all network sizes, the determinism-only experiment slows down when approaching complete convergence. This can be explained as follows. In these experiments nodes are initialized with a few random links all over the network, which are generally long-range links. Nodes that are priviledged to be initialized with links close to their target neighbohood, take a shortcut and converge very fast, replacing all their



(a) Bootstrapping of the entire network (b) Nodes joining a converged network

Fig. 7. Structure vs. Randomness in a torus topology. These graphs show the number of rounds it takes to reach the 99th percentile of convergence when bootstrapping an entire network (left), and the number of rounds for new nodes to join an already converged overlay (right). In all experiments, exactly 12 links are being exchanged by nodes when gossipping. Each line corresponds to a different network size (from 2^{10} at the bottom to 2^{17} at the top), and each dot corresponds to a different allocation of the 12 gossip slots to structured and randomized links.

initial random links with very specific, short ones. Soon enough, the network becomes nearly converged, and nearly all long-range links have been replaced by local ones. This, however, creates an obstacle to nodes that have not managed to converge yet, as they can only navigate slowly, in small local steps, towards their target neighborhoods, "crawling" in an almost converged overlay.

The aforementioned issue is circumvented by adding a "pinch" of randomness in an otherwise fully-deterministic configuration (fine blue line). This provides nodes with an extra source of random, potentially long-range, links. In accordance to our explanation in the previous paragraph, this visibly accelerates the last few stages of convergence.

Quite clearly, the balanced use of determinism and randomness (red line) outperforms all other configurations. This is a firm validation of our claim that both policies have distinct advantages to offer, which are best utilized in combination.

Determinism vs. Randomness Space Exploration Having developed an understanding on the specific roles of determinism and randomness in self-organization in a torus topology, we now run an extensive set of experiments to create a complete picture of their interaction.

We considered eight different network sizes, namely 2^{10} (1024), 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}, and 2^{17} (131072), and for each network size we considered all possible combinations of deterministic and randomized gossip lengths, so that the total gossip length stays fixed and equal to 12. This accounts to 13 experiments per network size, that is, all combinations such that $g_{str} \in [0,12]$ and $g_{rnd} = 12 - g_{str}$. For each experiment we recorded the number of rounds it took to establish 99% of the target links.

Figure 7(a) presents the results of these experiments. Each experiment is represented by a single dot, while dots corresponding to experiments on the same network size have been connected by lines. The lowest line corresponds to networks of 2^{10} nodes and the highest one to networks of 2^{17} nodes. The horizontal axis shows the specific combination of determinism and randomness used in each experiment. More specifically, the value on the horizontal axis corresponds to the g_{str} value of each configuration.

The first observation is that the dynamics of convergence follow the same patterns in all network sizes. It should be particularly noted that these results correspond to a single run per configuration, which prevents loss of information due to averaging.

The most distinguishing message from this graph is that the use of randomness alone (rightmost column) performs orders of magnitude worse than any other configuration. It can also be observed that the other extremem, that is, complete determinism (leftmost column) performs a bit worse than most other configurations that combine the two.

Node Joins In addition to the experiments carried out so far, where all nodes start the VICINITY protocol at the same time, we also want to explore the behavior of VICINITY when nodes join an already converged overlay.

We considered the same combinations of network sizes and protocol configurations as the ones of Figure 7(a). In each experiment, we first let the network converge to the target topology, and then we inserted a new node initialized with exactly *one* neighbor picked at random, and we recorded how many rounds it took for that node to find its target neighbors.

Figure 7(b) shows the number of rounds it took a node to reach its target vicinity in networks of the aforementioned sizes and configurations.

As expected, purely randomized views result in slower convergence. However, we observe remarkably bad behavior also for determinism-only configurations. The explanation is that, as has also been discussed earlier, navigating in an already converged overlay in the absence of random long-range shortcuts is a slow process.

These graphs emphasize our claim, that neither of the two policies is sufficiently good on its own. Determinism and randomness appear to be complementary in creating structure.

5.2 Clustering Nodes in Groups

Overview In this scenario, we assign each node a *group ID*, which constitutes its profile. The goal is to form clusters of nodes that share the same group IDs. From a node's perspective, the goal is to establish links to other nodes with the same group ID.

The only comparison operator defined on node profiles is equality of group IDs. By comparing their profiles, nodes can tell whether they belong to the same group or not. However, no other type of comparison or proximity metrics apply: any foreign group is "equally foreign", there is no notion of ranking or proximity. The target topology has been explicitly selected to form a non-connected, non-navigable graph, to shed light at the operation of VICINITY in such overlays.

The selection function $\text{SELECT}(p, \mathcal{D}, k)$ is simple and straightforward. It starts by selecting in a random sequence descriptors from $\mathcal{D}$ whose group ID is the same as

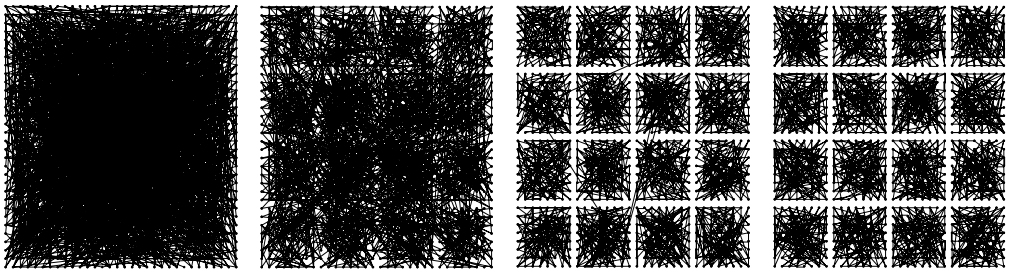


Fig. 8. Self-organization into 16 groups of 64 nodes each, in a 1024-node network.

p's. If these are fewer than k, it continues by selecting randomly from the rest of the descriptors.

Similarly to the torus scenario, Figure 8 provides a graphical illustration of an 1024-node network self-organizing into the target overlay. Again, nodes' deterministic and randomized views have been set to a size of six, each. Nodes are assigned group IDs such that a total of 16 groups exist, each having 64 nodes. Nodes are plotted in a layout that places members of the same group together, purely to make visualization intuitive. As far as the protocol operation is concerned, nodes do not have coordinates, but only their group ID. To avoid cluttering the graph, only two random outgoing links of each node's V_{str} view are shown, with links to foreign groups given higher priority. This way, when a node in Figure 8 appears to have no links to groups other than its own, it is guaranteed that *all* its V_{str} links point at nodes within its group.

Experimental Analysis Figure 9 presents the progress of self-organization of the grouping scenario, for the same network sizes and protocol settings used in the torus overlay. That is, network sizes of 2^{12}, 2^{14}, and 2^{16} have been considered, and the sum of the structured (g_{str}) and randomized (g_{rnd}) gossip length has been fixed to 12. Note that in this scenario, the group size is fixed to 64 nodes, therefore the networks of 4096, 16384, and 65536 nodes consist of 64, 256, and 1024 groups, respectively.

To better interpret the experimental results, we should build a good understanding of nodes' goals in this scenario. A node's task is divided in two steps: first, discover the right cluster; second, get well connected in it. The deterministic component of VICINITY excels in the second. Through a single link to the target cluster, a node rapidly learns and becomes known to additional nodes in that cluster. It turns out that the crucial step in this test case is the first one: discovering the target cluster.

Returning now to the results of Figure 9, the most important observation is that, contrary to the torus scenario, randomness is clearly the key component for self-organization. Determinism alone (solid blue line) is consistently *unable* to let nodes find their group partners, indefinitely failing to build the target topology.

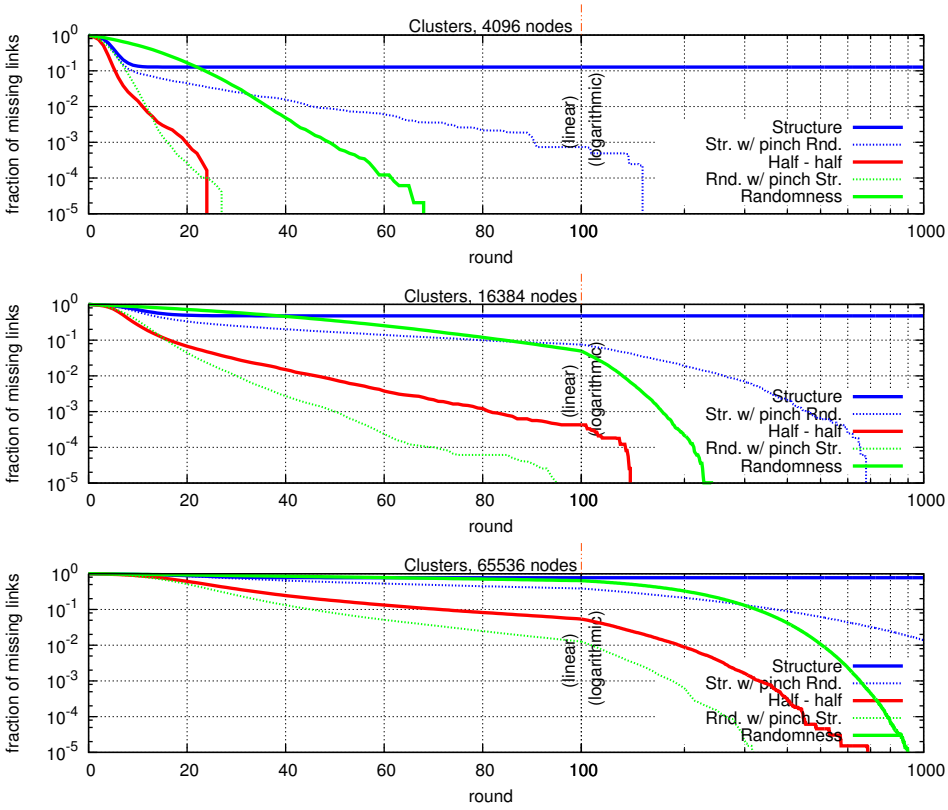
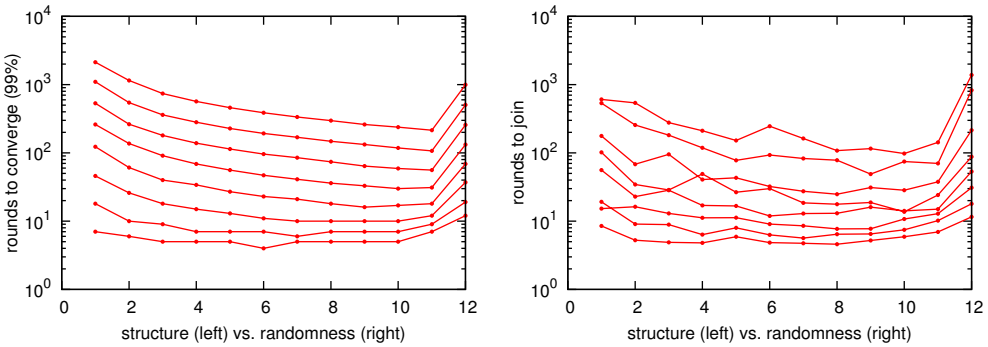


Fig. 9. Progress of self-organization in disjoint groups, for different configurations of VICINITY and a total gossip length (g_{str}+g_{rnd}) fixed to 12.

It is not hard to see why pure determinism fails. As nodes start clustering with other nodes of the same group, the pool of intergroup links in the network shrinks significantly. As explained above, once a node forms a link and gossips to one other node of its group, chances are it will acquire plenty of links to more nodes of the same group, rapidly trading its *intergroup* for *intragroup* links. In not so many rounds, most nodes end up having neighbors from their own groups exclusively. The problem comes with nodes that have not encountered nodes of their group early enough. If a node's neighbors are all from other groups, and these groups have already clustered into closed, self-contained clusters, the node has no chances whatsoever to be handed a link to a node of its own group, ever. A neighbor from such a self-contained foreign group can only provide alternative neighbors of that same, foreign group. The node, thus, finds itself in a dead end.

This demonstrates the need of a source of random, long-range links, to prevent such dead end scenarios. Indeed, just a "pinch" of randomness (fine blue line) is enough to save the day. It may not account for the most efficient configuration, but it clearly



(a) Bootstrapping of the entire network (b) Nodes joining a converged network

Fig. 10. Structure vs. Randomness in group clustering. The number of rounds it takes to reach the 99th percentile of convergence when bootstrapping an entire network (left), and the number of rounds for new nodes to join an already converged overlay (right). Experiments corresponding to value 0 of the horizontal axis do not converge, as they rely exclusively on determinism without any pinch of randomness.

bridges the huge gap between dead end and convergence. This is a particularly significant observation, as it clearly demonstrates that involving randomness, even just a "pinch" of it, is not just a matter of performance, but a matter of *correctness*.

When randomness acts on its own (solid green line), exposing each node to 12 random links in each round, convergence is certainly faster. However, in the lack of the deterministic component of VICINITY, a node should rely on randomness to discover *independently* each of the 12 target nodes of the same group.

Augmenting an almost complete randomness-based configuration with just a "pinch" of determinism (fine green line), gives the best achievable results. This was expected. Nodes, in this configuration, put nearly all of their communication quota on the hunt for same group nodes, through randomization. At the same time, this single link they reserve for targeted, deterministic communication, is sufficient to let them discover very fast all nodes of their group once they have discovered at least one of them.

Finally, the middleground configuration (red line), combining the deterministic and randomized components each with a gossip length of six descriptors, performs reasonably well in all cases, even if giving higher priority on randomness seems to improve things further for large networks.

Determinism vs. Randomness Space Exploration Similarly to the torus scenario, we perform a number of experiments to assess the performance of all combinations of determinism and randomness for a number of different network sizes.

Figure 10(a) plots the number of rounds needed for each experiment to build the target overlay. Recall that in the node grouping scenario, we identified randomness as being the key component for self-organization. This is clearly depicted in this graph, as the more randomness we use the faster we converge. However, when randomness is

used exclusively, without any assistance from determinism (rightmost column), convergence is slower.

Note that experiments corresponding to a determinism-only configuration (leftmost column) did *not* converge, hence they were omitted from the plots.

Node Joins Finally, we want to assess the number of rounds it takes new nodes to find their location in the target overlay, when joining an already converged network.

Figure 10(b) presents the results of these experiments. In accordance with the number of rounds it takes a whole network to converge from scratch, the rounds it takes nodes to join already converged overlays is very comparable.

The clear message from this graph is that, as we have consistently experienced also in our previous experiments, the two extremes should be avoided. Pure determinism in the case of node grouping, with a non-connected target structure, should be avoided by all means, as it will fail to build the target overlay. Pure randomness should also be avoided, as it will provide poor performance.

Concluding our entire evaluation of self-organization, we can state that picking a configuration that balances determinism with randomness, is a safe option for a system that self-organizes the network efficiently and works for diverse topologies.

6 Related Work

The work most closely related to VICINITY is the T-MAN protocol, by Jelasity et. al. [2, 3, 5]. T-MAN is focused exclusively on the deterministic structuring aspect in self-organization of overlays. Although its design does employ a peer sampling service, this is used exclusively for providing nodes with random views *once*, during intialization, as well as for synchronizing nodes to start the topology building process together. As such, it is targeted at *bootstrapping* overlays, rather than maintaining them under dynamic network conditions. For example, garbage collection for stale descriptors and support for nodes joining an already converged overlay have not been considered in the design. The baseline version of VICINITY, shown in Figure 2, is nearly equivalent to the T-MAN protocol.

Earlier efforts for self-organization of overlays have led to solutions that are tailormade for specific applications, such as [11], which clusters users of a file-sharing application based on the content they share.

BuddyCast is a file recommendation mechanism embedded in the Tribler [8] BitTorrent client. Inspired by [11], it essentially constitutes a deployment of VICINITY in the real world, clustering users by their file content preferences, to provide them with relevant recommendations.

7 Conclusions

Does randomness matter? The main conclusion from our research is a clear affirmative answer. In some cases, having probabilistic decision-making is even necessary.

In our study, we have concentrated exclusively on overlay construction and maintenance. For this domain it is also clear that structure matters as well. Having only randomness may severely affect the behavior of our overlay-maintenance algorithm. What is striking, however, is that adding either a pinch of randomness accompanying an otherwise deterministic technique, or adding a pinch of structure to an otherwise fully random process can have dramatic effects. In our examples we have been able to trace with reasonable confidence why such pinches of randomness or structure helped, but there is still much research to be done when it comes to developing more general insights and to identifying which classes algorithms and data structures benefit from randomness and which not.

The foundational question is why a *specific* mix of randomness and structure works so well, and how much of a pinch will indeed do the job. Our study sheds some light on this question, but also makes clear that much more work, and extended to other subfields, is necessary to come to a principled approach when dealing with designing large-scale distributed systems.

References

1. Fraser Cadger, Kevin Curran, Jose Santos, and Sandra Moffett. A survey of geographical routing in wireless ad-hoc networks. *IEEE Communications Surveys Tutorials*, 15(2):621–653, 2013.
2. Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Proceedings of Engineering Self-Organising Applications (ESOA'05)*, July 2005.
3. Márk Jelasity and Ozalp Babaoglu. T-Man: Gossip-based overlay topology management. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, editors, *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers*, volume 3910 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2006.
4. Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comp. Syst.*, 23(3):219–252, 2005.
5. Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Comput. Netw.*, 53(13):2321–2339, August 2009.
6. PeerNet. http://acropolis.cs.vu.nl/PeerNet.
7. PeerSim. http://peersim.sourceforge.net.
8. J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. Tribler: a social-based peer-to-peer system: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2):127–138, February 2008.
9. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.
10. Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.
11. Spyros Voulgaris and Maarten van Steen. Epidemic-style Management of Semantic Overlays for Content-Based Searching. In *EuroPar*, LNCS 3648, pages 1143–1152. Springer-Verlag, August/September 2005.
12. Duncan J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ, 1999.