# Capturing Complexity in Networked Systems Design: The Case for Improved Metrics

Sylvia Ratnasamy
Intel Research

## Abstract

*The systems and networking community lays great store by "clean", "elegant" system designs. Yet, our notion of what these terms mean often relies more on intuition and qualitative discussion than rigorous quantitative metrics. This paper questions whether we can do better and takes a first stab at quantifying this notion of complexity with regard to the algorithmic component of a networked system design.*

*While the success of our particular attempt is unclear, we believe identifying such metrics would be valuable not only in improving our own design and analysis methodologies but also to better articulate our design aesthetic to other communities that design for Internet contexts (*e.g.*, algorithms, formal distributed systems, graph theory).*

## 1 Introduction

The design of a networked system frequently includes a strong algorithmic design component. For example, solutions to a variety of problems – routing, distributed storage, multicast, name resolution, data processing in sensor networks, resource discovery, overlays – all define distributed procedures by which a collection of nodes accomplish a network-wide task.

A much valued property in Internet systems such as the above is that of design simplicity. However, as the literature reveals, our rationalization about the simplicity (or lack thereof) of design options is often through qualitative discussion or, at best, proof-of-concept implementation. What rigorous metrics we do employ tend to be borrowed from the theory of algorithms. These metrics however were intended to capture the overhead or efficiency of an algorithm and are at times incongruent with our notion of what makes for simple systems. For example, two of the most common metrics used to calibrate system designs are the amount of state maintained at nodes and the number of messages exchanged across nodes. However, most of us would consider flooding a simple although inefficient solution. Similarly, a piece of state obtained as the result of complex consensus or leader election protocol feels intuitively more complex than state that holds the IP address of a neighboring node.

We conjecture that this mismatch in design aesthetic contributes to the frequent disconnect between the more theoretical and applied research on networking problems. A good example of this is the work on routing. Routing solutions with small forwarding tables are widely viewed as desirable and the search for improved algorithms has been explored in multiple communities; *e.g.*, a fair fraction of the proceedings at STOC, FOCS, PODC and SPAA are devoted to routing problems. The basic distance-vector and link-state protocols incur high routing state ($O(n)$ entries) but are simple and widely employed. By contrast, a rich body of theoretical work has led to a suite of *compact* routing algorithms (*e.g.*, [1–4]). These algorithms construct optimally small routing tables ($O(\sqrt{n})$ entries) but appear more complex and have seen little adoption. Such discrepancies are even more common in the context of sensor networks where the difficulties of the operational environment render simplicity that much more valuable.

Note that this is not to suggest existing performance metrics aren't relevant or useful. On the contrary, all else being equal, solutions with less state or traffic overhead, are strictly more desirable. The point – or rather conjecture – here is that design simplicity plays a role in selecting solutions for real-world systems but existing performance-focused metrics can be incongruent with our notion of what constitutes elegant system design.

This paper raises the question of whether we can identify metrics that more directly capture the intuition behind our judgment of system designs. Some might view system design and evaluation as inherently reliant on the design aesthetic and experience of system designers. The conjecture behind this paper is that maybe this need not be true – the system designs we work with are sufficiently deterministic that there ought to be no fundamental reason why our appreciation of a design cannot be based on quantifiable measures. Such metrics, if we can identify them, would not only allow us to more rigorously discriminate between design options but also to better align the design goals of the algorithms and systems communities.

This paper takes a first stab at identifying such metrics. Our results are preliminary, intended primarily to initiate discussion on the merits and nature of alternate metrics. Moreover, we stress that our metrics are intended to complement, and not replace, existing performance metrics. For example, in the case of a routing algorithm, our metrics might capture the complexity of route construction but reveal little about the quality of computed paths. Finally, while we focus on system design at the algorithmic or procedural level, there are many aspects to a software system

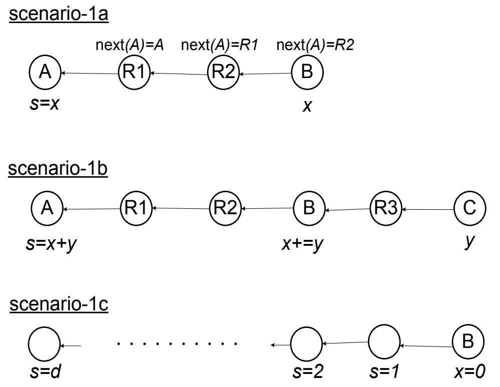

**Figure 1**: *Complexity in different scenarios.*

that contribute to its ultimate complexity. For example, as the CAP theorem [5] tells us, the very definition and prioritizing of a system's service model and guarantees, profoundly impacts complexity. The same is true for the sound design of its software implementation. Although at least as important as distributed complexity, these are not aspects we consider in this paper.

## 2 Strawman

At a high level, one might view much of system design as centered around the issue of *state* – defining what state is required, how it is constructed and used by different operations, and so forth. For example, a routing solution defines the forwarding entries required, the process by which nodes discover these entries, and how a packet is delivered end-to-end using this forwarding state. In all this, the strain particular to wide-area systems arises when state is distributed and hence a given piece of state is dependent not only on the different nodes storing its input state but also the network and intermediate nodes needed to relay this input state to the node in question. In other words, for a given piece of state, not only are its dependencies distributed, there are also more of them. Moreover, relative to a centralized or cluster environment, these dependency-inducing elements (input/relay nodes, links, *etc.*) tend to be more independent in their failure or change models. While traditional metrics count the amount of state but otherwise mostly treat all state as equal, we postulate that a key ingredient to capturing the difficulties in a networked system is to measure the *ensemble of distributed dependencies* that must hold together for a given piece of state to be consistent with the inputs from which it is derived.

In what follows, we attempt to develop such a metric. Metrics are only as useful as they are usable and, it is worth noting that current popular metrics simply count the total state and messages. These are conceptually simple, and lend themselves to evaluation through simple examination, analysis, or even mechanistically in simulation. A key goal we set is to define metrics that are somewhat similarly accessible. Our strategy – at least in this first cut – is to limit ourselves to metrics that only involve *counting* the different dependencies and avoid incorporating intricate models of node or link failure, state machine descriptions and the like. We discuss some of the limitations of our counting-based approach later in the paper.

We use a series of incremental observations and toy scenarios to help develop our proposed metric. Our discussion considers only distributed dependencies in state and, where the context is clear, abuses notation to let state identify the node storing the state; *e.g.*, instead of saying delivered to node $X$ that stores state $s$, we simply say delivered to $s$.

**Value *vs.* transport dependencies:** We start with the case where a piece of state, denoted $s$, is derived from a single input state, denoted $x$. For example, in scenario-1a in Figure 1, $x$ denotes the current temperature reading at node $B$ and state $s$ at node $A$ is assigned the value of $x$. The value of $s$ is derived from $x$ and hence any change in $x$ must be communicated to node $A$. By contrast, $s$ is dependent on the $next(A)$ state at $B$, $R1$ and $R2$ only for the delivery of $x$ to $s$ but a change in any of these does not require an update to $s$. We distinguish between these two forms of dependencies and say $s$ is *value* dependent on $x$ and *transport* dependent on $next(A)$, at B, R1 and R2.

Let $v_s$ denote the number of pieces of state on which $s$ is value dependent, and $t_{s\leftarrow x}$, the number of pieces of state relied on to transport $x$ to $s$. Since we're only interested in distributed dependencies, we set $v_s = t_{s\leftarrow s} = 0$ if $s$ was generated at the local node; thus, in Figure 1, $v_x = t_{x\leftarrow x} = 0$ and correspondingly, $v_s = 1$ and $t_{s\leftarrow x} = 3$. Note that a piece of state is not necessarily value dependent on its inputs. For example, say we defined $s$ as the temperature at node B at a specific time T1 (as opposed to B's current temperature). In this case, once established, $s$ is unaffected by changes at node B or the network between A and B. Thus, for $s$ derived from $x$, we set $v_s = v_x + 1$ if $s$ is value dependent on $x$ and $v_s = max(v_x, \varepsilon)$ otherwise, where $\varepsilon$ $(0 < \varepsilon \ll 1)$ is a minimal dependency value we introduce to ensure all non-local state has a non-zero dependency which also captures the one-time cost of state initialization. Similarly, to ensure than any inter-node communication incurs a minimal dependency cost, we set $t_{x\leftarrow y} = max(\varepsilon, t_{x\leftarrow y})$, for adjacent $x$ and $y$.

**Combining value and transport dependencies** Consider the slightly more involved scenario-1b in which $y$ records the current temperature at node $C$, $x$ represents the sum of $y$ and the current temperature at $B$, and $s$ is once again set to $x$. Now, $v_x = 1$ (since $x$ also depends on $y$) and hence $v_s = 2$. The transport dependency $t_{y\leftarrow y} = 0$, $t_{x\leftarrow y} = 2$, and $t_{s\leftarrow x} = 3$. We note that value dependencies accumulate in a fairly straightforward fashion but the extent or frequency with which transport dependencies $t_{s\leftarrow x}$ are incurred depends on the number of value dependencies downstream from $x$. For example, $s$ depends on $t_{s\leftarrow x}$

to relay changes in the temperature at either B or C but changes in $y$ are only relayed using $t_{x\leftarrow y}$.

Based on the above discussion, for state $s$ derived from a single input $x$, we define $c_s$, the complexity of $s$ as:

$$c_s = v_s \times t_{s\leftarrow x} + c_x$$

Thus, for $s$ in scenario-1, $c_s = 3$ and $c_x$ =0 while in scenario-2, $c_y$ =0, $c_x$=2 and $c_s = 2 \times 3 + 2 = 8$.

Note that $c_s$ emphasizes the simultaneous importance of balancing both value and transport dependencies in achieving low complexity – a single dependency input $x$ ($v_x$ =1) delivered to $s$ via a convoluted network path is deemed complex as is an input that is one hop away ($t_{s\leftarrow x} = 1$) but $x$ itself is derived from a long chain of previous inputs.

As a final example before proceeding, consider scenario-1c in Figure 1. Here $x = 0$ and each node computes $s$, its distance to $x$ by incrementing its right-hand neighbor's value of $s$ by 1. We abuse notation and let $d$ denote a node with $s = d$; then we have $v_d = d$ and $t_{d\leftarrow(d-1)} = 1$ and $c_d = d \times 1 + c_{d-1}$. We have $c_x = 0$ and hence $c_d = O(d^2)$.

**Multiple inputs** So far, we considered $s$ derived from a single input $x$. (Note that by input, we mean direct inputs; *e.g.*, in scenario-1b, we consider $x$ as input to $s$ but not $y$.) We now consider the case where $s$ is derived from $m$ inputs $x1$, $x2$, $\ldots$, $xm$. We consider two basic variants that can be combined to yield more complex scenarios. In the first, *all m* inputs are required to compute $s$ (*e.g.*, computing the min, max or average of $m$ input readings); in the second $s$ can be derived from *any* one of the $m$ inputs (*e.g.*, recording liveness). For simplicity, we assume $t_{s\leftarrow x_i} = 1$ in both cases.

When all $m$ inputs are required, we set:

$$v_s = \sum_{i=1}^{m} (v_{xi} + 1)$$

$$c_s = \sum_{i=1}^{m} ((v_{xi} + 1) \times t_{s\leftarrow xi} + c_{xi})$$

Thus if $v_{xi} = c_{xi} = 0$, we have $v_s = c_s = m$.

In the second scenario, $s$ can get by with any one of $m$ inputs coming through. Accordingly, we set the value dependency and complexity of $s$ as follows:

$$v_s = \frac{1}{m} \times \sum_{i=1}^{m} (v_{xi} + 1/m)$$

$$c_s = \frac{1}{m} \times \sum_{i=1}^{m} ((v_{xi} + 1/m)\, t_{s\leftarrow xi} + c_{xi})$$

Note that the above reflect the observation that in the one-of-$m$ variant, $s$ is less dependent on each individual input and does not depend on the sum total of all inputs. Again, when $v_{xi} = c_{xi} = 0$, we have $v_s = c_s = 1/m$.

| Case | Description | $v_s$ | $c_s$ |
|---|---|---|---|
| 1 | s=x; s,x are 1 hop apart | 1 | O(1) |
| 2 | s=x; s,x are *k* hops apart | 1 | O(*k*) |
| 3 | s=hops to x; s,x are *k* hops apart | k | O($k^2$) |
| 4 | s=ALL(x1,...,xm); s, xi 1 hop apart | *m* | O(*m*) |
| 5 | s=ANY(x1,...,xm); s, xi 1 hop apart | $1/m$ | O($1/m$) |
| 6 | s=x; *m* 1-hop paths from x to s | 1 | O($1/m$) |

**Table 1**: *The value dependency and complexity for a single piece of state s for various base-case scenarios.*

**Multiple paths** There may exist multiple paths by which an input can be delivered to the required node. For example, consider the case where $x$ is delivered to $s$ along any one of $m$ disjoint paths, and the transport dependency of each disjoint path is (say) $d$. We treat multiple paths akin to the corresponding multiple input scenarios and hence set $t_{s\leftarrow x} = d/m$ and hence the complexity $c_s = d/m$ which is lower than the single-input-single-path case by a factor $m$. [1]

Table 1 summarizes our complexity evaluation for state $s$ in the various toy scenarios. We see that, as one might expect, the complexity of state derived from a single input (case#1) is less than that derived from m inputs (#4) but greater than for one-of-m-inputs (#5). Similarly complexity decreases as the network offers more delivery options between input and output (1 vs. 6). Note too that, our metric penalizes a value dependency of $d$ that is accumulated in series or depth (#3) more than the same value dependency accumulated in breadth (# 4). This is reasonable as deeper dependencies incur more transport dependencies.

**Operations on state** So far we looked at computing the complexity of a given piece of state. A similar strategy can be used to compute the complexity of an operation – we treat the pieces of state the operation acts on as inputs and, based on how these inputs are combined, compute the operation's complexity from the individual state complexities. *E.g.*, a packet forwarding operation destined for D relies on the routing table entry for D at each of the series of nodes from the source to D. Specifically, recall the previous scenario-1c, where each node learns its distance and next hop to $x$ (node B). We had computed the complexity of state $d$ hops away as O($d^2$). Forwarding a packet from A to B requires the state at each intermediate node for a complexity of O($d^3$) (sum of squares). Or, consider a file download that takes one of $m$ inputs where each in-

[1] Many reviewers remarked that the decision to treat one-of-$m$ as having a factor $m$ lower complexity than the case of a single input/path is somewhat debatable because, ultimately, one of the inputs/paths is made use of. While this is a valid point that merits further scrutiny, the rationale behind the current choice is that a piece of state is less dependent on a single input if alternate inputs are easily available although this reasoning might be conflating simplicity and robustness. Note too that, while the complexity of a single piece of state (in the one-of-$m$ case) may be lower, the cost of creating more state for the purpose of redundancy will emerge in consider the net complexity of the complete system.

put is a pointer to a replica for the file. If each input has a complexity of (say) O($k$) then, akin to the one-of-$m$ inputs case, our download operation has complexity O($k/m$).

In the following section we present some preliminary analysis of more complete networked designs. However, before doing so, we discuss some of the limitations of our proposed metrics and possible improvements.

## 2.1 Limitations, Future Directions

**Correlated inputs:** Our formulation treats inputs as independent and hence might be over-counting the dependencies. For example, the above forwarding operation sums the state complexities at each hop even though these are related. The extent of inaccuracy this introduces as well as compensating measures remains to be studied.

**Discriminating between transport dependencies** : Our formulation merely counts the number of transport dependencies however each transport dependency is itself state with its own value dependency and complexity and taking these into account might lead to more discriminating metrics. For example, we might instead sum the value dependencies of each transport state.

**Capturing absent dependencies** Our formulation measures the complexity involved in having state be consistent with the inputs from which it is derived. However, this does not necessarily capture *all* the dependencies that cause the state to take the value it does. For example, we measured the complexity of finding the distance $k$ between two nodes $s$ and $x$. However, this value of $k$ depends as much on the *absence* of nodes between $s$ and $k$ that could lead to a different value of $k$ as it does on the presence of the $k-1$ nodes between $s$ and $x$.

**Robustness vs. Simplicity** Our formulation assigns lower complexity to state derived along alternate inputs/paths and hence reflects robustness to some extent. This link is however indirect and potentially limited; clearly relating complexity to robustness is an important future direction. Related is whether it might be useful to discriminate across inputs based on the degree to which the ouput (whether state or operation) depends on each input. For example, a DHT route critically depends on the successor entries but the absence of appropriate finger entries only leads to route degradation.

**Scope** Our metrics do little to validate the assumptions, correctness or quality of a solution. Capturing notions of consistency and convergence might require incorporating a notion of time or temporal dependency into our formulation and is another avenue for future exploration.

# 3 Initial Complexity Studies

This section presents preliminary analysis of a few common networked systems. We offer a high-level sketch of results with no detailed derivations; our intention is more to offer concrete examples of the type of analysis one might undertake in this context. We explore classical routing solutions in Section 3.1, and, in Section 3.2, look at resource location solutions in the context of P2P and sensor networks.

## 3.1 Network Routing

Our first study compares the complexity of distance-vector (DV) and link-state (LS) to the compact routing algorithm of Abraham *et al.* [4] (`AG+_compact`) which probably represents the state-of-the-art in compact routing. For simplicity, our analysis assumes a single shortest path to a destination.

In the case of DV, the routing entry (denoted $s$) for a destination $d$ hops away is akin to case-3 in Table 1 and hence $v_s = O(d)$ and $c_s = O(d^2)$ and an end-to-end forwarding operation has complexity $O(d^3)$. In LS, a node propagates its link information to every other node and hence the entry $e$ for a single edge has $v_e = O(1)$ and $c_e = O(d)$ (because the transport dependencies are O(d)). To compute the actual next-hop entry (denoted $s$) for a destination, LS requires the state for each of the $d$ edges to the destination and hence, once again, $v_s = O(d)$, $c_s = O(d^2)$ and end-to-end forwarding has complexity $O(d^3)$ akin to DV.

`AG+_compact` guarantees routing table sizes with O($\sqrt{n}$) entries and worst-case stretch no more than 3.0. Moreover, the stretch for Internet-like topologies has been shown to be ≈1.0 for Internet topologies [6], raising the question of whether compact routing might be an attractive option for IP routing. Briefly, `AG+_compact` operates as follows: a node A's vicinity ball (denoted VB(A)) is defined as the $k$ nodes closest to A. Node A maintains routing state for every node in its own vicinity ball as well as for every node B such that A ∈ VB(B). A distributed coloring scheme assigns every node one of $c$ colors. Under a slight relaxation this can be done by simply hashing the node name to a color. One color, say red, serves as the global backbone and every node in the network maintains routing state for all red nodes. Finally, a node must know how to route to every other node of the same color as itself. For $n$ nodes, vicinity balls of size $k = O(\sqrt{n}\log n)$ and $c = O(\sqrt{n})$ colors, one can show that a node's vicinity ball contains every color. With this construction, a node can always forward to a destination that is either in its own vicinity, is red, or is of the same color as the node itself. If none of these is true, the node forwards the packet to a node in its vicinity that is the same color as the destination. The challenge in `AG+_compact` lies in setting up routes between nodes of the same color without requiring state at intermediate nodes of a different color and yet maintaining bounded stretch for all paths. Loosely, `AG+_compact` achieves this as follows: say nodes A and D share the same color and A is looking to construct a routing entry to D. A explores every vicinity ball to which it belongs (VB(I), A ∈ VB(I)) and that touches or overlaps the vicinity ball of the destination D (*i.e.*, ∃ node X

∈ VB(I) with neighbor Y and Y ∈ VB(D)). For such C, A could route to D via C, X and Y. `AG+_compact` considers possible paths for each neighboring vicinity balls VB(C) as well as the path through the red node closest to D and uses the shortest of these for its routing entry to D. Discovering A's membership in a node B's VB itself incurs significant dependencies – unlike DV/LS where a node maintains distance for any and every unique destination it hears about, here B maintains state for A iff A is one of the k nodes closest to B. In other words, whether B maintains state for A depends on the relative distance of *other* nodes to B which already induces a dependency of at least O($\sqrt{n}$). Moreover, the construction of intra-color routing entries requires that A explore all vicinity balls in which it is contained, and those of each of the $O(\sqrt{n})$ like-colored nodes which yields a total dependency of $O(n)$ – significantly higher than DV or LS!

Such analysis offers hints for alternate designs. For example, we conjecture that one might reduce the above dependencies by $\sqrt{n}$ if we defined nodes' vicinity balls not as an ordering of nodes but in terms of the distance around each node; with this change, A's membership in B's vicinity ball would depend only on A, B and the nodes between them. While such a change would likely weaken the bounds on the size of routing tables it could offer lower complexity.

### 3.2 Resource Discovery

**P2P resource discovery** Many P2P applications locate resources using either unstructured (Gnutella) or structured (DHT) overlays. For the former, each node connects to some number of other peers and each neighbor entry $s$ thus has $v_s = 1$ and $c_s = 1$. (This assumes a transport dependency of 1 for overlay links.) By comparison, a DHT node might have $\log n$ neighbors, each with $v_s = 2$ and $c_s = 2\log n$ (due to a value dependency of one for a node's successor and hence two for a finger entry; the transport dependency is $\log n$ ignoring once again multiple paths). The corresponding complexity of end-to-end DHT routing is thus O($\log^2 n$).[2] This would seem to support deployment statistics and the common perception that unstructured solutions are simple, if inefficient. In absolute terms though, DHTs too exhibit low complexity which again would seem to concur with the general enthusiasm for DHTs in the systems and networking communities.

**Resource discovery in sensor networks** By far the most common approach to resource location in sensor nodes uses a flood-and-find approach where a sink floods the query over the entire network and relevant data is routed along the reverse path to the sink [7, 8]. The per-node network state here is merely the parent to the sink which, akin to the simple distance counting scenario in Section 2, has $v_s = k$ and $c_s = O(k^2)$ where $k$ is the node's distance to the sink. While $k$ and O($k^2$) appears fairly low complexity, it is worth noting that in a sensor network, $k$ can be O($\sqrt{n}$) leading to non-trivial complexity; we conjecture this may offer some insight on the engineering difficulties that have been reported for even simple tree construction [8, 9] and speculate that solutions based on gossip-style protocols [10] might be one approach to avoiding such scaling in dependencies.

To avoid the inefficiency of flooding, researchers have explored the use of in-network rendezvous mechanisms. One highly scalable proposal uses geographic addressing and routing [11–13]. In traditional geo routing, a node requires only the geo positions of its physical (*i.e.*, in radio range) neighbors. This incurs very low complexity – for each neighbor entry $s$, $v_s = 1$ and $c_s = 1 \times \varepsilon = \varepsilon$ (as discussed in Section 2, neighbor discovery effected through blind broadcasts might be viewed as incurring negligible transport dependency). Unfortunately, the adoption of geographic techniques has been hampered by both, concerns about the cost, power consumption and usability of GPS technology, and because empirical studies have repeatedly shown that wireless connectivity is not always congruent with geo proximity violating a core assumption of geo-routing. [3]

Two research directions address these concerns. Schemes such as CLDP [14] and GDSTR [15] continue to require GPS but propose novel route recovery algorithms that tolerate incongruities between physical distance and connectivity. The second approach eschews the use of geography altogether; schemes such as GEM [16] and NoGeo [17] instead construct virtual coordinate systems derived from only the measured connectivity between nodes. Like traditional geo routing algorithms, these new schemes are scalable in terms of the routing state but require additional mechanisms to either recover from route failures (CLDP, GDSTR) or to construct the virtual coordinate system (GEM, NoGeo). One might ask how much of the simplicity of traditional geo-routing is lost due to this? A quick analysis of the NoGeo protocol suggests that a routing entry $s$ has $v_s = O(n)$ and $c_s = O(n^{3/2})$ (due mostly to a periodic initialization phase to position O($\sqrt{n}$) perimeter nodes). While the various schemes should be explored in greater depth, the above suggests a significant increase relative to both flood-and-find and the idealized promise of geo routing.

## 4 DISCUSSION AND FUTURE STUDIES

Validating the goodness of a metric is, almost by definition, difficult and perhaps the best is to analyze a range of systems and examine the results. We close with a list of

[2] This DHT analysis may be overly generous as a node A's successor state is actually dependent not just on the identifier of A's successor but on the absence of any other node between A and its current successor; our metric does not currently capture such absent dependencies.

[3] Highlighting that complexity metrics do little to validate the assumptions behind a solution.

open questions and analyses that could help in this regard as well as offer insight on common design practices.

**Centralizing network computations: simpler?** Architectures that centralize the route computation have been proposed as a simpler alternative to today's distributed architecture [18] and it would be useful to undertake a formal analysis comparing the two. We conjecture the answer may depend on whether the value dependencies are "reset" at the centralized computation point. *I.e.*, on whether the final forwarding entries pushed to routers need to be consistent with the view of the world at the centralized route computation point or the true state of the world.

**Designing for low dependency** Section 2 presented a simple example where time-stamping temperature readings truncates the value dependency of the state being propagated. To some extent, soft-state protocols employ a somewhat similar strategy by bounding the lifetime of state and hence the length of dependencies it induces. Similarly, introducing redundancy in both inputs and transport dependencies lowers our measure of complexity. A useful exercise would be to quantify the complexity of systems that employ such techniques and verify whether their complexity matches our intuition.

**Layered *vs.* customized solutions** Some DHT applications [19–21] adhere to a standard DHT API and layer more complex functionality over this API while others [22, 23] choose to customize their DHTs to the task at hand. On the one hand, layering might lead to more needlessly inherited dependencies while the latter might introduce more mesh-like dependencies. In this context, one might for example compare the complexity of a CDN over a "sloppy DHT" interface [22] *vs.* the standard DHT interface or Mercury [23] that builds a customized solution to distributed range queries versus PHTs [21] that adopts a layered approach.

**Network addressing and routing options** A number of very different approaches to routing and addressing have been proposed in the context of both wireless and wired networks – gossip [10], synthetic coordinate systems [16, 17], clustering/dominating sets [24], tree-based [7,8], DHT-inspired [25] and so forth – that could be compared in terms of a more complexity-focused evaluation.

## 5 Acknowledgments

The author thanks Kevin Fall, Paul Francis and the anonymous reviewers for their valuable feedback.

## References

[1] L. J. Cowen. Compact routing with minimum stretch. In *ACM SODA*, 1999.

[2] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. Compact distributed data structures for adaptive routing. In *ACM STOC*, 1989.

[3] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *ACM SPAA*, 2001.

[4] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. In *16th ACM SPAA*, 2004.

[5] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent available partition-tolerant web services. In *SigACT News*, 2002.

[6] D. Krioukov, K. Fall, and X. Yang. Compact Routing on Internet-like Graphs. In *IEEE Infocom*, 2004.

[7] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: a scalable and robust communication paradigm for sensor networks. In *MOBICOM*, 2000.

[8] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad hoc sensor networks. In *OSDI*, 2002.

[9] Cheng Tien Ee, Sylvia Ratnasamy, and Scott Shenker. Practical data-centric storage. In *NSDI*, 2006.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickles: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI*, 2004.

[11] F. Kuhn, R. Wattenhofer, Y. Zhang, , and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *22nd ACM PODC*, 2003.

[12] B. Karp and H. T. Kung. Greedy Perimeter Stateless Routing for wireless networks. In *MOBICOM*, 2000.

[13] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks.

[14] Y. J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *NSDI*, 2005.

[15] B. Leong, B. Liskov, and R. Morris. Geographic routing without planarization. In *NSDI*, 2006.

[16] J. Newsome and D. Song. GEM: Graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of SenSys*, 2003.

[17] Ananth Rao, Sylvia Ratnasamy, Christos Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In *MOBICOM*, 2003.

[18] M. Caesar, D. Caldwell, N. Feamster, Jennifer Rexford, Aman Shikh, and J. Merwe. Design and Implementation of a Routing Control Platform. In *NSDI*, 2005.

[19] Matthew Harren, J. Hellerstein, Ryan Huebsch, B. Thau Loo, Scott Shenker, and Ion Stoica. Complex queries in DHT-based Peer-to-peer networks. In *IPTPS*, March 2002.

[20] F. Dabek et al. Wide-area cooperative storage with CFS. In *ACM SOSP*, October 2001.

[21] Yatin Chawathe et al. A case study in building layered DHT applications. In *Proceedings of SIGCOMM*, 2005.

[22] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In *NSDI*, 2004.

[23] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Support scalable multi-attribute range queries. In *SIGCOMM*, 2004.

[24] J. Gao, L.J.Guibas, J .Hershberger, L. Zhang, and An Zhu. Discrete mobile centers. In *Proceedings of the Symposium on Computational Geometry*, 2001.

[25] Matthew Caesar *et al.* Virtual Ring Routing: Network routing inspired by DHTs. In *SIGCOMM*, 2006.