The Promise, and Limitations, of Gossip Protocols

Ken Birman (ken@cs.cornell.edu)
Dept. of Computer Science, Cornell University

Abstract

Recent years have seen a surge of interest in gossip protocols, with proposals to apply them for purposes ranging from autonomic self-management, repair of inconsistencies, reliable multicast and distributed search. Yet the field of distributed computing is littered with technologies that had initial promise, but were ultimately rejected by the industry. Researchers who measure their work through its impact need to ask some tough, basic questions. What are the uses to which gossip is particularly well-matched, and what are its limitations? What alternatives are there to gossip-based solutions, and when would we be better-off using a non-gossip protocol? When, in effect, is gossip the technology of choice?

Gossip 101

In this white paper, we will consider a protocol to be a gossip protocol if it satisfies the following conditions:

1. The core of the protocol involves periodic, pairwise, inter-process interactions
2. The information exchanged during these interactions is of (small) bounded size
3. When nodes interact, the state of one or both changes in a way that reflects the state of the other. For example, if A pings B just to measure the round-trip-time between them, it isn't a gossip interaction.
4. Reliable communication is not assumed
5. The frequency of the interactions is low compared to typical message latencies, so that the protocol costs are negligible
6. There is some form of randomness in the peer selection. Peer selection might occur within the full node set, or might be performed in a smaller set of neighbors.

It is useful to distinguish three styles of gossip protocol [3].

1. *Dissemination (rumor-mongering) protocols.* These use gossip to spread information; they basically work by flooding nodes in the network, but in a manner that produces bounded worst-case loads:
   a) *Event dissemination protocols* use gossip to carry out multicasts. They report events, but the events don't actually trigger the gossip (since gossip runs periodically). One concern here is the potentially high latency from when the event occurs until it is delivered.
   b) *Background data dissemination protocols* continuously gossip about information associated with the participating nodes. Typically, propagation latency isn't a concern, perhaps because the information in question changes slowly or there is no significant penalty for acting upon slightly stale data.
2. *Anti-entropy protocols for repairing replicated data,* which operate by comparing replicas and reconciling differences.
3. *Protocols that compute aggregates, or that accomplish some task as a side-effect of computing an aggregate.* These operate by sampling information at the nodes in the network and combining the values to arrive at a system-wide value – for example, the worst case load on any node in the system, or the best match with some search pattern [9].

Aggregation is a surprisingly complex topic. The key requirement is that an aggregate must be computable by fixed-size pairwise information exchanges; these typically terminate after a number of rounds of information exchange logarithmic in the system size, by which time an all-to-all information flow pattern will have been established. Aggregators can be based on simple operators such as max or union, but one can also design probabilistic aggregators that converge rapidly towards average, sum, etc [13], and will be exact provided that no failures occur while the protocol is running. An interesting scenario arises in the Astrolabe system [20], where gossip is used to construct an overlay tree, and then aggregation is performed purely using gossip, but with a pattern of peering determined by the tree. Astrolabe can easily compute exact sums, count elements satisfying conditions, etc. Many kinds of aggregators depend upon having an estimate of the system size, but several protocols Gossip protocols for size estimation include [14][15].

In some cases, the main reason for computing an aggregate is to accomplish some form of side-effect. For example, many overlay tree construction algorithms can be reformulated as gossip aggregation algorithms (as in the T-Man project [8] [9]). Similarly, there are gossip protocols that can arrange the nodes in a gossip overlay into a list sorted by node-id (or some other attribute) in logarithmic time using aggregation-style information exchanges that effectively carry out a parallel exchange sort [10]. The use of an underlying gossip infrastructure to support higher-level abstractions is further explored in [7] and [15].

This is a rather inclusive definition, and some might argue that it is too broad. In particular:

1. Many protocols that predate the earliest use of the term "gossip" fall within our definition. Many routing protocols have an underlying gossip-like information exchange. In the view of the author, the fact that the term "gossip" arose after protocols were already using some of these ideas is not particularly disturbing. Indeed, an interesting question arises: what was the first use of gossip in a distributed system? The answer is not apparent.
2. We've used the term "gossip aggregation" in conjunction with ways of using a tree-like structure while computing aggregates. Astrolabe doesn't actually aggregate "on a tree": in the system, the tree is purely an abstraction used to assign each node its role (or roles) in the the aggregation algorithm. The movement of data is purely by peer-to-peer gossip using random peer selection. But one can also use gossip to construct an overlay tree, then aggregate within the tree, sending data from the leaves towards the root. While this author believes that Astrolabe fits our definition, a case can be made that this second form of aggregation does not.

[1] ken@cs.cornell.edu. This work was supported in part by grants from AFRL/IFSE, AFOSR, NSF and Intel Corporation.

Notice that a gossip substrate can easily implement a standard routed network: nodes could "gossip" about traditional point-to-point messages, effectively tunneling normal traffic through the gossip layer. Bandwidth permitting, a gossip system can potentially support any classic protocol or implement any classical distributed service. Nonetheless, when we talk of gossip protocols, we rarely intend such a broadly inclusive interpretation. More typically we have in mind protocols that run in a regular, periodic, relatively lazy, symmetric and decentralized manner; the high degree of symmetry among nodes is particularly characteristic. Thus, to give a simple example, while one could run a 2-phase commit protocol over a gossip substrate, piggybacking the messages on gossip traffic, doing so would be at odds with the spirit, if not the wording, of the definition.

Frequently, the most useful gossip protocols turn out to be those with exponentially rapid convergence towards a state that "emerges" with probability 1.0. For example, a classic distributed computing problem involves building some form of tree whose inner nodes are the nodes in a network, and whose edges represent links between nodes (for routing, as a dissemination overlay, etc). Not all tree-building protocols are gossip protocols (for example, spanning tree constructions in which a leader initiates a flood), but gossip offers a good decentralized solution that can be useful in many situations.

Some researchers have begun to use the term *convergently consistent* to describe protocols that achieve exponentially rapid spread of information and, therefore, converge exponentially quickly to a globally consistent state after a new event occurs, in the absence of additional events. For this purpose, a protocol must propagate any new information to all nodes that will be affected by the information within time logarithmic in the size of the system (the "mixing time" must be O(log(N))).

An interesting and apparently open question concerns the "need" for this form of exponential convergence. When a gossip protocol becomes overloaded (by an excessive event rate), or experiences certain patterns of correlated failure, one can imagine an information mixing process that might be slower than O(log(N)). Indeed, one could imagine deliberately designing gossip protocols that have slow mixing rates. Such a protocol would exhibit a slower than exponential convergence property, but is this necessarily a bad thing?

The Limitations of Gossip

The foregoing discussion should make it clear that gossip, in one sense, is no more or less limited than any other distributed computing paradigm. Yet the stylized manner in which we normally use gossip introduces significant limitations.

First, consider the implications of the small, bounded message sizes and the relatively slow periodic message exchanges. These combine to severely limit the information carrying capacity of a gossip algorithm. For example, if gossip is used to disseminate information (often, in a form of flooding), the system-wide capacity for new events will be limited simply because the aggregate "bandwidth" available is bounded. If we consider that a typical gossip multicast resides in the system for O(log(n)) time, during which it keeps every infected node busy infecting other nodes the average rate at which new messages can be sent, system-wide, will be roughly the inverse of the residency time: 1/log(n). In typical protocols, a node gossips about any given piece of information for *f* rounds (the fanout of the gossip protocol), after which it becomes quiescent, at least with respect to that information. Often *f* is set to log(N).

Notice that because messages have a fixed maximum size, if the amount of information that a node needs to gossip exceeds the maximum carrying capacity of messages, the effective fanout will start to fall because nodes will be incapable of gossiping about some information items during the expected number of rounds. This illustrates a potentially subtle issue: while gossip protocols scale well in some dimensions, there are senses in which these protocols might not scale well at all. In the case just described, we can see that a steadily increasing rate of events can exhaust the carrying capacity of the gossip information channel, and our protocol may then malfunction. The precise point at which this saturation occurs depends on many factors: the rate at which events enter the system, event sizes, gossip fanout and message sizes, and the nature of the gossip protocol itself. Aggregation, for example, should be relatively immune to such issues, while event dissemination may be highly sensitive to it.

This should concern us, because it would be natural to assume that in a system of N nodes, the rate of new events will be proportional to N. Our observation makes it clear that the rate at which new events can be introduced can only rise to some system-wide threshold, after which the rate of introducing new events will need to be *reduced,* potentially in proportion to N! On the other hand, bursty event rates that briefly overwhelm a gossip protocol may not be a problem, provided that the fanout is large enough so that the protocol will still be gossiping about the excess events when the channel load finally starts to fall. In this sense, we can see that the real issue is the information carrying capacity of gossip as a rate over time. Quantification of this issue is clearly needed.

The relatively slow rate of gossip can be an obstacle. While it is common to claim that users need only tune the gossip rate to match their goals, requirement 5 complicates the picture. To a point, one can reduce the periodicity to speed the spread of information at the cost of increased overheads. However, eventually this tactic results in a gossip rate that approaches the network RTT. Were one to go to such an extreme, resource contention effects at the network interface and within the network itself might distort the behavior of the protocol, and many of the most salient characteristics of gossip would be lost.

In his talk at the Leiden workshop (the basis of this special issue of OSR), Van Renesse has observed that gossip is not a particularly robust protocol, particularly with respect to correlated loss patterns and to malicious behaviors (components that malfunction, for example by running the protocol incorrectly, disseminating incorrect data, and so forth). For example, suppose that we are trying to find the five nodes in a system that are detecting the highest rate of "possible intrusion attempts". A natural approach would be to ask nodes to gossip about this rate, and also to use aggregation to gossip about the five highest rates that they have heard about. Now, suppose that we implement this approach, but some malicious or malfunctioning nodes report falsely high rates, or retain the minimum rather than the maximum when comparing intrusion rates. Such errors can delay or even defeat the protocol. Clearly, gossip is very much a community process: all the nodes are in some sense dependent upon the correct behavior of all other nodes. Thus, while gossip is often characterized as being "highly robust", this is really limited to certain classes of failures.

Strengths of Gossip

The limitations just cited are just one side of a complex picture. Gossip protocols also have substantial power. Among the most cited strengths are these:

- *Convergent consistency*. As mentioned earlier, this refers to the fact that properly designed gossip protocols, when not overwhelmed by a higher rate of incoming "events" than the information-carrying bandwidth of the underlying channels, should have a logarithmic mixing time. Typically, this implies that if we compare the states of nodes with respect to information that entered the system at various times in the past, nodes will agree strongly on the set of events older than logarithmic time units in the past (the unit of time being the gossip round length), and that any inconsistencies (differences in the event sets) will be limited to recent events.

  Notice that because of the bounded gossip rates and bounded message sizes, most gossip protocols can "fall behind" if a system is exposed to a surge of incoming events. Designers typically parameterize gossip protocols so that such situations will be rare and will not persist for long.

- *Emergent structure*. Earlier, we contrasted a classic deterministic protocol for building a spanning tree by leader-initiated flooding with a decentralized way of building such a tree, for example as done in the T-Man system [8]. In the gossip style, the tree "emerges" from randomized pairwise interactions between peers. An *emergent structure* is intended to evoke the image of a data structure that will be convergently consistent; although the structure would often have at best a probabilistic guarantee of satisfying the relevant structural invariant, it would often do so with probability 1.0 if one waits long enough while the system is quiescent. The structure may then continue to evolve over time as further gossip occurs.

- *Simplicity*. Most (but not all) gossip protocols are extremely simple. Often, gossip protocols require just a few lines of code and are completely symmetric: every node runs the identical code.

- *Bounded load on participants*. Many classic (non-gossip) distributed protocols are criticized because they can generate high surge loads that overload individual components. Gossip is normally used in ways that produce strictly bounded worst-case loads on each component, eliminating the risk of disruptive load surges. In some situations, where network capacity is also a concern, peer-selection is further biased to control load imposed on network links. Indeed, not only does a well-designed gossip protocol typically have bounded loads, these loads are often almost neglible – a tiny fraction of available bandwidth.

- *Topology independence.* If running on a sufficiently connected networking substrate, and with sufficient bandwidth, a gossip protocol will often operate correctly on a great variety of underlying topologies. Of course, some topologies are inadequately connected (the theory of expander graphs is applicable here: to be "adequately" connected, a protocol should trace out an expander graph [18][19]); when gossip runs on one of these defective graphs, the outcome of the protocol may be affected, either by a longer running time or through the manifestation of erroneous outcomes, such as logical partitioning (see Allavena [1]). Kempe has suggested that the theory of Small Worlds graphs might be used to design gossip overlays on which information will propagate at an optimal rate [12].

- *Ease of local information discovery.* Many gossip protocols are used for purposes of discovery, for example to find a nearby resource (these are usually protocols in which gossip occurs between neighbors, not between arbitrarily distant peers). Unlike local flooding, which scales poorly, gossip would typically find local information less quickly but with bounded costs: perhaps, a constant or a delay logarithmic in the system size.

- *Robustness to transient network disruptions.* As time elapses, there are exponentially many routes by which information can flow from its source to its destinations. However, as noted earlier, it is important to keep in mind that not all uses of gossip are robust in all ways. For example, unless data is self-verifying, dissemination protocols are often vulnerable to data corruption. Anti-entropy protocols may similarly be at risk of a replica becomes corrupted. Correlated failures can be an issue (network partitioning being the most extreme example); these can selectively delay the propagation of information, perhaps long enough so that nodes cease to gossip about it, leaving a persistent inconsistency. And aggregation protocols are vulnerable not just to the introduction of faulty information, but also to computational errors that result in a faulty computation of the aggregate.

Appropriate roles for gossip

Earlier we observed that gossip can be used to "do" anything one might wish to do in a distributed system: a gossip system can run a classical algorithm. The converse also holds: a gossip solution to a problem will invariably be just one of several ways to solve that problem. Before deciding to use gossip as the basis for an algorithm, we need to evaluate the options and should pick the best solution, or mixture of solutions.

For example, a primary use of gossip is for information diffusion: some event occurs, and our goal is to spread the word. Gossip offers a style of flooding that can reliably solve this problem with especially little needed code. Moreover, the reliability level can be very high; unless malicious faults are considered, probability 1.0 outcomes are common. Yet a significant criticism of this type of protocol is that the running times can be slow and that the protocol is potentially costly in terms of messages used. Gossip is a particularly bad way to implement a scalable event notification service, unless one cares little about efficiency or delay and places an exceptionally high value on some other attribute of the gossip solution, such as its ability to operate in networks with irregular and unknown connectivity.

If we compare a gossip diffusion multicast with a classical reliable multicast protocol, the latter will often approximate an ideal pattern in which each message reaches destination nodes exactly once. If properly designed, the reliability mechanisms can be extremely fast, and it is often possible to aggregate multiple small messages into larger ones for higher efficiency, and to exploit IP multicast or other technologies if available. A classical reliable multicast could potentially send millions of similarly sized messages in the same amount of time as it might take a gossip-flooding algorithm to propagate just a small fraction of those number of messages.

We need to be careful when making such comparisons: the classical algorithm can potentially use all the bandwidth in the network, while the gossip algorithm has a very small and bounded communication cost. But *even if the classical algorithm was* *constrained to use identical bandwidth to the gossip version, the classical algorithm will outperform gossip.* The gossip protocol has no "urgency" and much of its bandwidth is consumed by redundant information.

This overhead, and the delay associated with the periodicity of gossip rounds, ensures that no matter how fast we run a gossip protocol, it won't compete favorably if latency or "goodput" is compared with the best reliable multicast options.

The foregoing observation isn't necessarily damning. The classical algorithm may require careful configuration; the gossip solution potentially avoids this. Classical reliable multicast protocols can be complex, hence one might favor a gossip solution for its simplicity. But once a protocol is coded and debugged, it is just a tool, like any other tool. Once a protocol is installed and running, configuration may not seem like such an issue. The fact that gossip can be coded easily doesn't make it a fundamentally better tool than some other protocol that can't be coded quite so easily. Moreover, the randomness inherent in many gossip protocols can make it hard to reproduce and debug unexpected problems that arise at runtime, whereas many multicast protocols support strong models (virtual synchrony, state-machine styles of consensus, transactional 1SR) that can be helpful both as design tools and debugging aides.

Thus one should be wary of arguing that simpler protocols are better than more complex ones. As Einstein famously remarked, a thing should be as simple as possible – but no simpler. If the user actually *needs* a strong consistency model such as the ones cited, it would be unfortunate to instead offer a weaker probabilistic tool. Simplicity is a virtue, but not an overriding one.

One could offer similar comments about aggregation or anti-entropy applications. Gossip is certainly a convenient communication pattern for certain kinds of aggregation, but if we have a suitable overlay tree available, it will often be far faster to aggregate by sending a request to start the process down the tree, and then collecting a wave of data up the tree from leaves to root, aggregating at inner nodes. Gossip can be used to reconcile divergence between replicas, but just one of many possible options – and as mentioned in our discussion of faults, one must also pause to ask why replicas might become inconsistent in the first place. An anti-entropy repair mechanism can only help if some form of omission is at the heart of the inconsistencies we're trying to remedy. If the problem is actually some form of malicious fault, gossip is not helpful: it may actually provide a route whereby an intruder who has compromised one node can indirectly compromise others by tricking the gossip protocol into copying incorrect data to correct nodes.

The Road Ahead

What should be the uses of gossip in future systems? In the foregoing, we touched upon (some) merits and weaknesses of gossip. In the remainder of this short essay, our hope is to draw some conclusions based on the observations made above. At the risk of overgeneralizing, gossip stands out as the "best" solution for at least some purposes, each of which raises interesting opportunities for future study.

- Gossip can be a powerful adjunct to a classical protocol, for example when a gossip mechanism is "married" to some other technology that suffers from infrequent defects of its own. For example, suppose that we implement a classic style of information replication or configuration management as part of a data center, optimizing the protocols to squeeze every last cycle out of the hardware. It is common for systems built this way to exhibit various forms of fragility. A gossip solution could be constructed to run side-by-side with the high speed solution. This gossip mechanism may be slow but will also be predictable and reliable, and can be used to monitor the underlying infrastructure. The extreme robustness against communication disruptions, coupled with the certainty that the technology will never produce any kind of disruptive load bursts of its own (and hence won't exacerbate whatever problem we're trying to detect and repair) makes gossip a good choice. The gossip mechanism can give the application developer confidence that if a problem does arise, it can be corrected and repair initiated within a fairly short delay – here, a few seconds may seem like an appealingly rapid reaction time. Moreover, the simplicity of the gossip solution avoids the classic dilemma of a failure handling mechanism that turns out to malfunction in ways that introduce additional failure modes.

  An example of a gossip-based system that operates in this manner, as an adjunct to other technologies, arises at one of the major e-tailers in the context of their data centers. That company uses the Astrolabe [20] system for self-configuration, problem detection and repair.

  A second example of this sort of pairing arose in designing the Bimodal Multicast protocol and the closely related lpbcast protocol [2][4]. In these cases, an unreliable multicast protocol is paired with an out-of-band gossip mechanism to achieve high reliability and very good scalability.

- As a simple and rapidly adaptive way to construct overlay structures (see, for example, [8][17]). Many classical algorithms for building non-trivial structures (such as the trees used in certain DHTs) have later been found to have difficulty coping with churn or transient partitioning, high overheads, poor scalability, and so forth. Moreover, proving them correct is often very hard. A probabilistic proof that a gossip-based algorithm for discovery of an emergent data structure will quickly converge (with probability 1.0) is frequently almost trivial.

- Gossip opens the door to a variety of self-organizing and self-adaptive (autonomic) behaviors. In organizations concerned about the total cost of owning and operating a system, these properties are attractive. For example, Montressor has shown that gossip can be used to place servers or other sorts of super-peers in a manner that will situate a server close to each client, and will create enough servers to balance load [11], a problem related to utility placement. While Montressor's scheme is heuristic, it does suggest a path whereby results from the optimization community might be imported into distributed systems and architected to operate in an autonomous manner.

- Gossip can sometimes pair in a convenient way with some underlying physical form of locality (e.g. when nodes gossip with neighbors to discover nearby information). See, for example [21]. The ability to write topology-indifferent code that will be slow but robust can be extremely appealing if the discovery of the topology might otherwise be difficult or require some form of manual "help" in the form of a hand-coded configuration aid.

- Gossip offers an extremely decentralized form of information discovery, and its latencies are often acceptable if the information won't actually be used immediately. For example, in settings where one wants to support decentralized search without collecting information into any form of centralized store, a gossip protocol may be a good solution. On the negative side, humans expect snappy response, and any situation in which the end-user might experience a delay until logarithmically many rounds of a gossip protocol have run will probably be unacceptably slow, even if rounds are executed fairly quickly.

  This is a good context to recall that gossip is often dramatically outperformed by simple classical techniques, such as aggregation on a tree – the same observation applies to search, which can be understood as a directed form of aggregation. As we observed earlier, using gossip to build an overlay tree, within which we could run a 2-phase search or aggregation algorithm, may yield a far better solution than trying to rely upon gossip as the sole story. In particular, notice that if we use log(N) time to build an overlay tree and then use the tree to compute a function such as sum, median, or even the number of nodes, we obtain an accurate answer in log time. In contrast, we noted earlier that a pure gossip algorithm for solving such problems will typically yield only a probabilistic approximation to the correct answer.

- Gossip has interesting parallels to all sorts of social structures. For example, as individuals carrying wireless devices move about and interact, there may be opportunities to pass information among the devices (we have in mind examples such as a rapidly responsive traffic monitoring application, for example, that might exchange gossip between automobiles). Humans make heavy use of physical forms of peer-to-peer interaction technologies (cell phones, chat, automobiles that share the same roadways) and gossip is a natural technical substrate for building applications that might have a structure "mirroring" the physical one. Indeed, if the relevant data could be captured, gossip could be used to pursue some questions of great contemporary interest. This connection, and the connection to work on so-called Delay Tolerant Networks, would be well worth exploring.

The above are all examples that contemplate the use of gossip as a tool in what might be characterized as "classic" distributed systems – systems of the sort we are familiar with from the Internet and Web. But one can also speculate more ambitiously.

For example, For example, consider the difficulty epidemiologists have tracking outbreaks of viruses such as SARS, e-coli or avian flu. If we could build simple gossip-based mechanisms to track the patterns of interactions and contact between residents of a city, or a rural community (by watching for peering opportunities when their cell telephones come within range), it might be possible to create much better models for use in predicting the way that future pandemics will spread. Indeed (flipping the question around), one could use gossip between small applets of this sort to "mimic" the spread of a virus, and then compare the outcomes with what would be predicted by current models, as a way of validating the models. Here the value of gossip is that the technology is so extremely simple, and after all, is out there in the field with peering opportunities in very much the same situations where humans could potentially infect one-another. The domain of gossip as a research tool for experimental sociology thus strikes us a particularly promising one (provided that such issues as privacy could be addressed).

It is easy to imagine a future in which these kinds of completely novel uses of gossip could turn out to be the "killer applications" for the technology. After all, despite the obstacles involved in doing so, classical ways of building classical styles of distributed computing system are well accepted within the field. Applications such as the one just described, in contrast, are completely outside of current day-to-day experience.

Towards real systems

Developers of real systems face basic choices as to the technology they will use to build higher level abstractions. These choices have very basic implications when the developer constructs the lowest layers of a system: a system that will communicate over TCP links leads to a completely different internal architecture than one that will employ home-brew protocols that use gossip-style interactions over UDP, or one that will make use of an outboard reliable multicast protocol running in a library or a separate daemon. Our analysis makes it clear that gossip could be an ideal choice for some purposes, and for systems designed (only) for those purposes, the developer's path is relatively clear.

But what should developers do if they confront a complex problem, for which some mixture of technologies will be needed? Today, there is no obvious answer to such a question. Scalability makes this more than just a software engineering and systems architecture question: if a collection of applications shares a platform and will run over a common communication substrate, there may be many simultaneous but quasi-independent uses of gossip, of multicast, or of TCP. Should a platform treat these separately, or might there be strong reasons to aggregate traffic or other functionality at the operating systems communication layer? The author sees these as pressing problems that could reveal a completely unnoticed scalability "landscape" for the gossip research community to explore.

Indeed, the broader question implied by these musings seems to be perhaps the most interesting one of all. How should communication solutions be embedded into modern platforms, to maximize utility in such dimensions as scalability, but also ease of application development, efficient sharing of the platform among multiple applications that happen to be running on it, ease of debugging, of system deployment, and of long term aspects such as management, problem diagnosis and repair? How, in effect, can tools such as gossip be made part of the standard repertoire of technologies available to developers? The gossip community has enjoyed a period of relatively low-hanging fruit, during which all sorts of clever uses of gossip were demonstrated. But now the technology faces a basic problem of maturity: to have impact, gossip needs to learn to integrate with alternatives, and with the prevailing operating systems and development architectures. Happily, doing so exposes a fascinating research agenda.

Conclusions

Our discussion can be summarized in a few rules of thumb:

- Gossip is a tool, not an end in itself. It should be used selectively, in settings where gossip is the best choice.
- This implies that gossip needs to "learn" to coexist with other solutions in the context of general-purpose platforms.
- Gossip-based algorithms that discover structures are an area of particularly high value.
- Gossip is *necessarily slow* and shouldn't try to compete where speed is the goal.

- Some of the most exciting future opportunities arise where gossip is employed in a setting that has some sort of inherent resonance with a peer-to-peer style of communication, and that can't even be approached with classical protocols.

Acknowledgements
The author wishes to acknowledge the suggestions of the anonymous referees, Anne-Marie Kermarrec, Robbert van Renesse, Einar Vollset and Ingrid Jansch-Porto.

References

[1] **Correctness of a Gossip-based Membership Protocol.** André Allavena, Alan Demers and John Hopcroft. Proc. 24th ACM Symposium on the Principle of Distributed Computing (PODC 2005).

[2] **Bimodal Multicast.** Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu and Yaron Minsky. ACM Transactions on Computer Systems, Vol. 17, No. 2, pp 41-88, May, 1999.

[3] **Epidemic algorithms for replicated database management.** Alan Demers, et. al. Proc. 6th ACM PODC, Vancouver BC, 1987.

[4] **Lightweight probabilistic broadcast.** Patrick Eugster, Rashid Guerraoui, S. B. Handurukande, Petr Kouznetsov, Anne-Marie Kermarrec. ACM Transactions on Computer Systems (TOCS) 21:4, Nov 2003.

[5] **Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead.** Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse. Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)

[6] **Systematic Design of P2P Technologies for Distributed Systems.** Indranil Gupta, Global Data Management, eds: R. Baldoni, G. Cortese, F. Davide and A. Melpignano, 2006.

[7] **Efficient and Adaptive Epidemic-Style Protocols for Reliable and Scalable Multicast**. Indranil Gupta, Ayalvadi J. Ganesh, Anne-Marie Kermarrec. IEEE Transactions on Parallel and Distributed Systems, vol. 17, no. 7, pp. 593-605, July, 2006.

[8] **T-Man: Gossip-based overlay topology management.** Márk Jelasity and Ozalp Babaoglu. Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Springer-Verlag LNCS 3910 (2006).

[9] **Gossip-based aggregation in large dynamic networks.** Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. ACM Transactions on Computer Systems, 23(3):219–252, August 2005.

[10] **Ordered slicing of very large overlay networks.** Márk Jelasity and Anne-Marie Kermarrec. IEEE P2P, 2006.

[11] **Proximity-aware superpeer overlay topologies.** Gian Paolo Jesi, Alberto Montresor, and Ozalp Babaoglu. Proc SelfMan 06. Spinger-Verlag LNCS 399, Dublin, Ireland, June 2006.

[12] **Spatial gossip and resource location protocols.** David Kempe, Jon Kleinberg, Alan Demers. Journal of the ACM (JACM) 51: 6 (Nov 2004).

[13] **Gossip-Based Computation of Aggregate Information.** David Kempe, Alin Dobra, Johannes Gehrke. Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS). 2003.

[14] **Active and Passive Techniques for Group Size Estimation in Large-Scale and Dynamic Distributed Systems.** Dionysios Kostoulas , Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, Al Demers. Elsevier Journal of Systems and Software, 2007.

[15] **Build One, Get One Free: Leveraging the Coexistence of Multiple P2P Overlay Networks**. Balasubramaneyam Maniymaran, Marin Bertier and Anne-Marie Kermarrec. Proc. ICDCS, June 2007.

[16] **Peer counting and sampling in overlay networks: random walk methods.** Laurent Massoulié, Erwan Le Merrer, nne-Marie Kermarrec, Ayalvadi Ganesh. Proc. 25th ACM PODC. Denver, 2006.

[17] **Chord on Demand.** Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Proc. 5th Conference on Peer-to-Peer Computing (P2P), Konstanz, Germany, August 2005.

[18] **Introduction to Expander Graphs.** Michael Nielsen. http://www.qinfo.org/people/nielsen/blog/archive/notes/expander_graphs.pdf. Technical report, June 2005.

[19] **Building low-diameter P2P networks**. G. Pandurangan, P. Raghavan, Eli Upfal. In *Proceedings of the 42nd Symposium on Foundations of Computer Science* (FOCS), 2001.

[20] **Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining.** Robbert van Renesse, Kenneth Birman and Werner Vogels. ACM Transactions on Computer Systems (TOCS) 21:2, May 2003.

[21] **Exploiting Semantic Proximity in Peer-to-peer Content Searching.** S. Voulgaris, A.-M. Kermarrec, L. Massoulie, M. van Steen. Proc. 10th Int'l Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004), Suzhou, China, May 2004.