How robust are gossip-based communication protocols?

Lorenzo Alvisi
Laboratory for Advanced Systems Research (LASR)
Dept. of Computer Sciences
The University of Texas at Austin

Jeroen Doumen
Distributed and Embedded Systems Group
Dept. of Computer Science
University of Twente

Rachid Guerraoui
School of Computer & Communication Sciences
Swiss Federal Institute of Technology in Lausanne (EPFL)

Boris Koldehofe
Institute of Parallel and Distributed Systems (IPVS)
University of Stuttgart

Harry Li
Laboratory for Advanced Systems Research (LASR)
Dept. of Computer Sciences
The University of Texas at Austin

Robbert van Renesse
Dept. of Computer Science
Cornell University

Gilles Tredan
IRISA/INRIA-Rennes

ABSTRACT

Gossip-based communication protocols are often touted as being robust. Not surprisingly, such a claim relies on assumptions under which gossip protocols are supposed to operate. In this paper, we discuss and in some cases expose some of these assumptions and discuss how sensitive the robustness of gossip is to these assumptions. This analysis gives rise to a collection of new research challenges.

Categories and Subject Descriptors

C.2.4 [**Computer Communication**]: Distributed Systems; D.2.4 [**Operating Systems**]: Security and Protection

Keywords

gossiping, security, incentives, robustness

1. INTRODUCTION

Gossip (or epidemic) communication protocols are an attractive technique to manage inconsistencies that arise in distributed systems because, allegedly, they are simple to implement and highly resilient to failures. A replicated database can converge to a consistent state using a gossip protocol, despite temporary partitions and process failures [5]. Messages can be broadcast with high reliability and steady throughput despite high network packet loss rates and high process failure probabilities [4].

Since the early work on database replication and multicast, many have considered gossip protocols and touted their robustness. The robustness claim is often backed-up by analyses accounting for message loss, process failures, topology changes, and so on. However, as we shall see, they also rely on a set of additional assumptions about the environment in which the protocols operate.

In this paper, we take a close look at these assumptions and at the role they play in determining gossip's robustness. We discover, somewhat disturbingly, that gossip's ability to deliver on its promise of robustness depends crucially on a handful of assumptions that are often left unspoken and, as such, have so far largely escaped close scrutiny.

Our findings suggest that robustness is far from an established property of gossip protocols—rather, it is an exciting open research challenge. We conclude this document by articulating some promising directions for further investigation.

2. BACKGROUND

In a simple gossip protocol, each process periodically and randomly selects a partner with whom it exchanges recently observed information. Gossip's robustness stems from this random communication pattern that allows processes to route new information around both communication and process failures. Much like real-life rumor- and virus-spreading, information disseminated by a gossip protocol spreads quickly and reliably with high probability [7].

In their pioneering work, Demers et al. use a gossip-based protocol to resolve inconsistencies among Clearinghouse database servers [5]. In each communication round, a database replica randomly selects other replicas with whom it gossips about recent database updates. This straightforward protocol guarantees eventual consistency, that is, in the absence

of further updates all replicas converge to the same database state with probability 1.

Birman et al. take a gossip-based approach towards recovering from message loss in multicast [4]. In each gossip round, instead of gossiping about database entries, processes exchange recent multicast receipt events. Such events are tagged with a maximum hop count chosen large enough so that gossip guarantees that each event reaches every process with high probability [8]. At the heart of the dissemination procedure lies the ability for a process to select a communication partner, randomly, from the entire set of processes.

Having global membership is a strong assumption, and this assumption was revisited in [6]. In this protocol, the very knowledge of membership is also gossiped around. Several variations of gossip-based membership protocols have been studied [26, 15, 12, 1, 27].

Gossip protocols can be adapted to tolerate a certain number of process crashes by adjusting the maximum hop count and fan-out parameters (the number of selected partners in each communication round). These techniques impact the convergence rate of the protocol [3, 14, 23, 17]. In DRUM [2], Badishi et al. use resource bounding and port hopping techniques to thwart denial-of-service attacks.

Most gossip protocols assume crash failures. This assumption can be eliminated by creating protocols that tolerate Byzantine failures. Malkhi et al. initiated the study of gossip-style update diffusion in a Byzantine environment [20]. Their analysis was fundamentally different from previous ones because correct processes needed to verify the veracity of an update *without* using digital signatures. Later, Malkhi et al. [21] and Minsky et al. [22] independently improved upon this work by annotating propagation paths onto updates.

Later works on Byzantine gossip protocols differ from these first efforts by relying on cryptographic primitives. Johansen et al. designed Fireflies [13], a Byzantine fault-tolerant membership management protocol that allows processes to maintain a reasonably current view of the system's members. Haridasan and Van Renesse implement SecureStream [11], a live streaming system built on top of Fireflies. Concurrently, Li et al. designed BAR Gossip [19], a live streaming protocol that combines elements from the Byzantine fault-tolerance literature and the game theory literature to tolerate Byzantine and rational processes.

A common technique towards weakening assumptions in network protocols is to design them so that they adapt to changing environments. For example, Rodrigues et al. present a gossip-based broadcast protocol that automatically adjusts the per-node emission rate to a level that the gossip protocol can maintain without overflowing message buffers [24]. Previous gossip protocols tacitly assumed that the rate of emission is sufficiently low to avoid such congestion. In order to increase the rate of emissions Kouznetsov et al. [18] proposed age-based purging of message buffers. Subsequent work [16] also analyzed appropriate buffer sizes depending on the emission rate and evaluated buffer sizes in combination with different purging strategies.

3. HIDDEN ASSUMPTIONS IN GOSSIP PROTOCOLS

Any protocol makes certain assumptions about the environment in which it is deployed in order to guarantee certain properties. These include assumptions about network latency and bandwidth, processing time, failures, and so on. If such assumptions are made explicit, then the burden lies on the deployer to make sure that the assumptions are satisfied. However, in this section we present a set of five commonly made assumptions that are typically *not* made explicit. Yet, gossip protocols rely on these assumptions just as much as they rely on explicit ones.

ASSUMPTION 3.1. *In a gossip protocol, participants gossip with one or more partners at fixed time intervals.*

This assumption is technically a *synchrony* assumption. That is, the clocks at the different processes have the same rate of progress, and the processes gossip in real-time and are not arbitrarily slow. In addition, while most protocols allow for message loss, it is assumed that a certain percentage of messages is delivered within a round, and thus the network is also assumed to be synchronous.

In practice, neither the processes nor the network can be assumed to be synchronous, and arbitrary delays happen regularly. If, for example, gossip intervals were made very short compared to network latencies and CPU processing times, then most published analyses of gossip would certainly not apply. On the other hand, increasing this delay impacts the latency of gossip dissemination. Even with a generous gossip interval unfounded synchrony assumptions could lead to unexpected problems, such as unexpected bursts of message loss of delay that invalidate the analysis of delivery probability and/or latency.

A synchrony assumption can even purposely be invalidated by a malicious Denial-of-Service attack, either at the network level, by increasing network latencies, or at the application level, by creating large amounts of information to be gossiped.

ASSUMPTION 3.2. *There is a bound on how many updates are concurrently propagated.*

This assumption relates to the amount of resources that processes need to provide and may impact gossip's scalability. Most gossip protocols assume that processes have sufficient resources available and that gossip never exceeds these limits—hence, they typically do not consider what happens to a gossip protocol whose demands grow to require more resources than many of the participants are willing to budget for.

For instance, if each process generates new updates at a fixed rate, then, to keep track of the updates that have been already applied, each process will have to allocate buffers whose size grows linearly with the number of processes in the system.

Even if we assume there are sufficiently many resources available for storing information, it is hard to ignore gossip's communication costs. Too many messages may reduce the available bandwidth and increase latencies. There is a relation to Assumption 3.1. If the rate at which information is injected into the system exceeds the capacity of gossip, dissemination delays will continuously grow. Moreover, buffer exhaustion may lead to losing the memory of old events: in an asynchronous system, these events could then be mistaken for new updates and applied repeatedly.

Although it can be unacceptable for many applications, such behavior can easily occur if some processes slightly diverge from their expected execution steps. In particular, in a system where communication partners change dynamically it is hard to distinguish between processes which are malicious and processes that simply do not have accurate time. Processes can overload the system by sending messages at a slightly higher rate than they are expected to sent.

ASSUMPTION 3.3. *Every gossip interaction is independent of concurrent gossiping between other processes.*

In reality, gossip messages are transfered over the underlying physical communication graph. As a result, various communication failures are often correlated, but most analyses of gossip assume that message loss failures are independently distributed (if not identically). In practice, a single link failure could result in a large amount of dependent message loss.

Even in the absence of link failures, loss is often caused by queue overflows at routers, and indeed random gossiping between end-hosts could result in a highly uneven load on routers in the physical network. Such uneven load can also be caused by external traffic, which can either be of an unintended or malicious nature.

Finally, there is often a considerable amount of heterogeneity in real distributed systems. Not all processes are the same, not all links are the same, and not all routers are the same. In the face of these real-world difficulties, a mathematical analysis is often intractable. Simulations, on the other hand, may not give all the insights necessary to predict how gossip will behave when subject to a particular environment.

ASSUMPTION 3.4. *Any two processes can discover each other independently of the gossip mechanism.*

It is clear that any gossip protocol needs some kind of bootstrap mechanism whereby newly joining processes can discover one another. However, it is perhaps not obvious that this mechanism has to be available continuously. Gossip is often touted to work well in the face of transient network partitions. If a network is partitioned, each section can make progress independently through gossip. When the partition resolves, gossip can repair extant inconsistencies. We pose that making this work satisfactorily in practice depends on a discovery service.

To wit, consider a system with a large number of processes. Because of a network failure, a site with a few processes is disconnected from the rest of the membership. Now consider what will happen within this site. As each process chooses its gossip partners randomly, most attempts at gossip will fail and dissemination of information among these few processes will take exceedingly long.

It is therefore necessary that these processes may refer back to a discovery service in order to rediscover each other. At the same time, for efficiency it is important that the processes cease trying to gossip with the partners they can no longer reach. But herein lies a problem as well. If they do so and the network partition heals subsequently, gossip by itself may fail to rejoin the disconnected processes. Again, the system relies on a discovery service to reconnect the processes.

ASSUMPTION 3.5. *Processes select gossip partners within a round in an unpredictable random-like fashion.*

Random partner selection seems to be an essential aspect of gossip protocols. If this selection is not random then an application could be adversely affected. For instance, the mixing time could be slowed down considerably, some processes may receive messages less reliably than other processes, or processes may become partitioned. In some environments, using a suitable pseudo-random generator is sufficient for partner selection. In settings in which processes behave selfishly, however, there may be incentive for a process to not use such a generator.

Consider a selfish process S interested in the data from process T. Nothing prevents S from initiating gossip with T each round to get early access to T's data, and subsequently, S may be uninterested in gossiping with other neighbors. When processes select partners in an unregulated way, we lose our ability to guarantee traditional properties like logarithmic mixing time and resilience to failures.

Discussion. We do not claim that the above list of assumptions is comprehensive. There are often other hidden but obvious assumptions in any protocol. For example, if processes choose partners uniformly at random, memory overheads on processes would grow linearly in the number of processes. While this overhead matters little on current desktop machines, the memory on nodes in a sensor network could be severely limited. The point is that for every gossip deployment, we need to revisit our assumptions, and justify them in the appropriate light.

4. CHALLENGES

By revisiting the hidden assumptions of Section 3 we now propose research challenges in maintaining, and in some cases restoring, gossip's robustness properties.

- Assumption 3.1 states that many gossip protocols rely on synchrony assumptions. A natural evolution of research in distributed systems is to understand possibilities and impossibilities if we move to a weaker system

model. For example, research on consensus protocols has studied the impact if we move from a synchronous to an asynchronous system model, or if we take a different failure model into account. This way it was possible to establish many important research results. For instance, in the context of consensus it is impossible to guarantee termination, even for a single crash failure. This could also be natural direction for gossip protocols towards a better understanding of the guarantees we can hope for when changing the system model. What are the failures we could cope with if we apply gossip in an asynchronous model relying on fair links. Is it still possible to guarantee that dissemination of updates reach every correct process with probability 1?

- Revisiting Assumption 3.2 highlights a difficulty of resource constraint approaches. In the context of event dissemination a fairly small increase in the message rate can be sufficient to overload buffers and, even worse, overload the entire event dissemination system such that latencies grow without bounds. As proposed by Rodrigues et al. [24], one could use adaptation in order to control the emission rate of events for all processes. An alternative approach to enforce expected behavior would be to establish some resource management scheme which grants that only a particular set of processes can access the resources. Decentralized and dynamic resource management schemes to support scalability were discussed by Gidenstam et al. [10]. Would this also be an option for gossip protocols where only a few processes have a token to add new information, whilst the others can only listen to new updates? Could we use resource management to establish alternative ways to address consistency for applications? Can we ensure that the resources are managed decentralized in a fair manner and what happens if we need to manage resources when processes behave selfishly or otherwise maliciously?

- In Assumption 3.3 we find that in many cases the selection of gossip partners show dependencies with respect to the underlying communication graph, that is, failures could be correlated and communication delays may be unevenly distributed. However, it is still possible to set up logical but partially connected communication meshes on which it is possible to gossip successfully. How should those meshes be designed so that interactions between the logical links are minimized? Alternatively, can Internet routers support the illusion of independent gossips using something similar, say, to Random Early Detection (RED) [9]? Can we guaranteethat all members are treated fairly, and none are starved for delivery.

- Looking at Assumption 3.4 we can see that discovery is an important part of a deployed gossip mechanism. Still there does not exist much research devoted towards this issue. It would appear that a system addressing discovery should be aware of the physical topology of the network, and sufficiently decentralized in order to make sure that every node can access the discovery service at any time. While such a service may use gossip internally (for example, Astrolabe comes close to such a system [25]), it would seem that there is a chicken-and-egg problem.

- Many have taken gossip as a scalable and robust way to disseminate information in a peer-to-peer environment. Yet, at Internet scale in which each machine corresponds to a user and many users are selfish (witness file-sharing activity), we need ways to enforce our partner selection algorithms. Unfortunately, it is not obvious how to check that a process is selecting partners appropriately because this selection is often nondeterministic. Li et al. [19] enforce proper partner selection in their gossip protocol by using cryptographically generated choices. Their algorithm allows the recipient of a gossip request to check that the sender selected appropriately, and if not, to reject the request. A limitation of Li et al.'s approach is that the checking step relies on all processes sharing the same static global membership list. Is it possible to combine elements from Li et al.'s approach with the large body of literature on maintaining partial membership lists? Furthermore, can partner selections be biased in checkable ways to account for changing network topologies?

5. CONCLUSIONS

Like many other protocols, gossip protocols rely on a set of assumptions. Exposing these assumptions is crucial to understanding the robustness of the protocols that rely on them. We reviewed several assumptions made by gossip protocols, and explored the impact of invalidating the assumptions. This in turn suggested various research directions.

6. REFERENCES

[1] A. Allavena, A. Demers, and J. E. Hopcroft. Correctness of a gossip based membership protocol. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing (PODC '05)*, pages 292–301. ACM Press, 2005.

[2] G. Badishi, I. Keidar, and A. Sasson. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. *IEEE Transactions on Dependable and Secure Computing*, 3(1):45, 2006.

[3] N. T. J. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Griffin, 2nd edition, 1975.

[4] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[5] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 1–12. ACM Press, 1987.

[6] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *ACM Transaction on Computer Systems*, 21(4):341 – 374, 2003.

[7] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[8] P. T. Eugster, R. Guerraoui, and P. Kouznetsov. D-reliable broadcast: A probabilistic measure of broadcast reliability. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March, Hachioji, Tokyo, Japan*, pages 636–643, 2004.

[9] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[10] A. Gidenstam, B. Koldehofe, M. Papatriantafilou, and P. Tsigas. Dynamic and fault-tolerant cluster management. In *Proceedings of the Fith IEEE International Conference on Peer-to-Peer Computing*, pages 237–244. IEEE, Aug. 2005.

[11] M. Haridasan and R. van Renesse. Defense against intrusion in a live streaming multicast system. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P '06)*, pages 185–192, Washington, DC, USA, 2006. IEEE Computer Society.

[12] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proceedings of the ACM/IFIP/USENIX 5th International Middleware Conference*, pages 79 – 98, 2004.

[13] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable support for intrusion-tolerant overlay networks. In W. Zwaenepoel, editor, *Proceedings of Eurosys 2006*. ACM European Chapter, April 2006.

[14] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 565–574. IEEE Computer Society Press, Nov. 2000.

[15] A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, Mar. 2003.

[16] B. Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS '03)*, pages 76–85. IEEE, Oct. 2003.

[17] B. Koldehofe. Simple gossiping with balls and bins. *Studia Informatica Universalis*, 3(1):43–60, 2004.

[18] P. Kouznetsov, R. Guerraoui, S. B. Handurukande, and A.-M. Kermarrec. Reducing noise in gossip-based reliable broadcast. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems (SRDS '01)*, pages 186–189. IEEE, Oct. 2001.

[19] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proceedings of the 2006 USENIX Operating Systems Design and Implementation (OSDI'06)*, Nov. 2006.

[20] D. Malkhi, Y. Mansour, and M. K. Reiter. On diffusing updates in a byzantine environment. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS '99)*, page 134, Washington, DC, USA, 1999. IEEE Computer Society.

[21] D. Malkhi, E. Pavlov, and Y. Sella. Optimal unconditional information diffusion. In *Proceedings of the 15th International Conference on Distributed Computing (DISC '01)*, pages 63–77, London, UK, 2001. Springer-Verlag.

[22] Y. M. Minsky and F. B. Schneider. Tolerating malicious gossip. *Distributed Computing*, 16(1):49–68, 2003.

[23] B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, Feb. 1987.

[24] L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, and A. Kermarrec. Adaptive gossip-based broadcast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, San Francisco, CA, June 2003.

[25] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed systems monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(3), May 2003.

[26] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-based failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, pages 55–70, England, September 1998.

[27] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.