A Generic Theoretical Framework for Modeling Gossip-Based Algorithms

Yaacov Fernandess
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel
fery@cs.huji.ac.il

Antonio Fernández
LADyR, GSyC
Universidad Rey Juan Carlos
28933 Móstoles, Spain
anto@gsyc.escet.urjc.es

Maxime Monod
School of Computer & Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
Maxime.Monod@epfl.ch

1. INTRODUCTION

During the past 30 years of the Internet revolution, the Internet has become a major force of change with an enormous effect on civilization. Consequently, computer networks have evolved into more complex system and become virtually ubiquitous. This in turn, has given raise to a growing demand for scalable and reliable computer system architectures. Thus far, there has been enormous effort by the research community to introduce decentralized, simple, and scalable distributed systems to solve a wide range of problems. In this paper we explore one promising solution, which was initially inspired by mathematical models that investigate two everyday life phenomena, epidemics and gossip, which we used interchangeably throughout the paper. During the last century, mathematicians developed models to predict the rate of diseases spread, namely epidemics, using differential equations. In addition, researchers have developed discrete mathematics models to predict what we already know; rumors spread fast, namely gossip. It was thus natural to harness these models in order to design distributed systems that mimic the basic behavior of such fast spreading everyday life paradigms.

To begin with, gossiping and broadcasting were first described as two different information dissemination problems for a group of individuals connected by a communication network. More concretely, as defined in [14], in the gossiping problem, every individual in the network initially knows a unique item of information and needs to communicate it to everyone else in the network, whereas the broadcasting problem is defined as the case in which one individual has an item of information which needs to be communicated to every other individual. The authors present solutions for both problems, producing a deterministic sequence of unordered pairs of communication partners. Each pair represents a *phone call* made between a pair of individuals, such that, during each call, the two people involved exchange all the information they know at that time. At the end of the sequence of calls, everybody knows all the information that had to be spread. The survey focuses on the number of calls among n people over arbitrary network topologies and variants of the problems.

A similar version of the gossiping problem has been studied in parallel processing (see, e.g., [10, 17, 27, 33, 34]). As in [14], initially there is a collection of nodes, each holding a packet. The nodes are interconnected via a network of a given fixed topology and the goal is eventually to have every node obtain a copy of each packet. The communication is assumed to be parallel and synchronous, following a systolic model [24]. The main reasons to consider gossiping in this model are its conceptual simplicity and its potential to efficiently solve a wide range of parallel processing problems.

Lately, a new variety of gossip-based algorithms have evolved as communication patterns for designing simple, scalable, and efficient communication protocols in large distributed systems. To the best of the authors' knowledge, the first work proposing such gossip-based algorithms is due to Demers *et al.* [5]. This paper proposes a family of gossip-based algorithms for maintaining replicated database systems. Since then, gossip-based algorithms have been proposed to solve central problems in numerous distributed systems deployed over wide range of physical networks. These problems include, for instance, replicated database maintenance [5], Usenet news distribution [25], implementation of e-mail distributed systems [3], ad-hoc routing [13], distributed failure detection [36], network management [35], lightweight broadcast [7], peer-to-peer membership maintenance [11, 16, 37], aggregation in sensor networks [6], building of overlay structure [12], and topology management [15].

Position and Objectives. Gossip-based algorithms are distributed algorithms, in which nodes take steps and exchange information in execution cycles in order to provide a solution to given problems in a distributed manner. In this paper we factor out the fundamental elements found at the heart of these algorithms, hopefully, leading to a better understanding of this innovative form of communication.

Thus far, many of the gossip-based algorithms proposed in the literature have been evaluated empirically, and not theoretically. We believe that this is due to limitations of the existing theoretical models in modeling real distributed environments, on which gossip-based algorithms are deployed. That is, most existing theoretical models (that have been used to derive various theoretical results) do not capture all the key factors to which real life distributed systems are exposed. Consider, for an example, the dynamic behavior and scale of peer-to-peer networks, in which information and network's topology are changing continuously over time. These networks require robust mechanisms for keeping nodes updated with new content. Gossip-based algorithms have been used to spread information in such dynamic environments. However, thus far, theoretical models assume a static communication network topology.

Additionally, theoretical analysis of gossip-based algorithms will not only serve to evaluate the properties of these algorithms but also to uncover their strengths and weaknesses compared to traditional distributed algorithms. Nevertheless, in order to conduct such theoretical analysis, we need a general model, which captures the main characteristics of the distributed environment in which we choose to deploy. In this paper we try to face this problem, by defining a generic framework that can be used as starting point to model a distributed system in which one chooses to deploy an arbitrary gossip-based algorithm.

We factor out key elements found at the heart of gossip-based algorithms:

Problem A distributed computing problem that the gossip-based algorithm solves.

System The communication network (e.g., wired, wireless, overlay, internet) and environment in which we choose to deploy the algorithm.

Efficiency The goal is to design efficient algorithms, which is manifested in the following criteria:

- **Time complexity** Total number of rounds for delivery, as measured from the start state until termination.
- **Message complexity** The total amount of data transferred over the network channels throughout an execution.
- **Connectivity complexity** The total number of communication channels established throughout an execution.
- **Space complexity** The total amount of memory devoted to the algorithm throughout an execution.

1.1 Organization

The rest of the paper is organized as follows. The elements of the framework are presented and described in Section 2. Section 3 gives a formal definition of termination and several complexity measures that are of interest in the analysis of gossip-based algorithms. Finally, conclusions and future directions are given in Section 4.

2. GENERIC FRAMEWORK

A gossip-based algorithm, as we understand it in this paper, provides a *service* which is the solution to a problem (e.g., broadcasting information, random peer sampling, computing some aggregate value, etc.). The given algorithm solves the problem on top of an underlying system that provides itself some basic services (typically communication primitives) and some guarantees (e.g., connectivity). The link between the service and the underlying system is the algorithm itself, which is in charge of using the services provided by the underlying system to provide the desired service. In this section we describe these three main elements around which a gossip-based algorithm is built.

2.1 Problem Definition

Any gossip-based algorithm solves a problem in a distributed system in which there is a set V of n processes or nodes, each having a unique identifier and with communication capacity between them. For the definition of the problem, which is the topic of this section, we do not impose any specific mean of communication. Although it is commonly assumed that communication is via message passing, other forms, like shared memory, could also be possible. The distributed system considered could be dynamic in its composition. This means that nodes can be in at least two states, *active* or *inactive*, and that at different times different sets of nodes can be active. We denote by $V(t)$ the set of active nodes at time t and define $V = \bigcup_t V(t)$.

2.1.1 Problem Classification

Theoretical work on gossip-based algorithm has mainly focused on three classes of problems: *information spread*, *aggregate computation*, and *overlay management*.

Information Spread Given a state in which one of the nodes, $v \in V$ (called the source node), has a message m_v, the objective is that eventually every node has a copy of message m_v [5, 7, 9, 18, 20, 28].

Computing Aggregates Given a state in which each node $v \in V$ has a value x_v, the problem is to compute some aggregate function f (e.g., SUM, AVERAGE, MIN, or MAX) of all the values, x_v, so that every node eventually has the aggregate value [4, 19, 28].

Overlay Management In the overlay management problem (e.g., [1, 7, 11, 37]) each node $v \in V$ has a subset $view_v \subseteq V$, such that v has direct communication with every node in $view_v$. The nodes in $view_v$ are the neighbors of v in a overlay network OV. The objective is to achieve an overlay network with some property P (e.g., that each set $view_v$ is a uniform random sample of V).

Unlike the previous two classes, overlay management is usually a *continuous problem*, in the sense that it is not enough that the property P is reached, but it also must be maintained afterwards. This will make it different from the other problems classes.

2.1.2 Algorithm Termination or Problem Solution

In principle, identifying when a solution of one of the above problems has been found seems simple. However, that usually implies a very strict definition of the problem solution. We propose here alternative definitions of problem solution or algorithm termination, in the line of [9].

For instance, a strict definition of termination for information spreading algorithms is very natural: an arbitrary information spreading algorithm $\mathcal{A}$ terminates when all nodes have received a copy of the message m_v that is being spread. Unfortunately, even the "best" algorithm may never be able to guarantee termination as defined, due to the dynamics of the network, the churn, or the random behavior of the algorithm itself. For this, it seems convenient to introduce an approximation factor $\delta \geq 1$ to the complete dissemination of the message. Let $V(t)$ be the set of nodes in the system at time t, and $M_v(t)$ denote the subset of these nodes at that time that have the message m_v. We say that the spreading algorithm $\mathcal{A}$ has terminated at time t if the following predicate is satisfied

$$Term_{\mathcal{A}}(t) = \left\{ \frac{|V(t)|}{|M_v(t)|} \leq \delta \right\}.$$

A similar issue appears in gossip-based algorithms that compute aggregates. Let us assume that the algorithm $\mathcal{A}$ computes the aggregate function $f : R^n \mapsto R$ over the input values x_{v_i}, $v_i \in V$, where $V = \bigcup_t V(t)$. Let $\widehat{f}_{v_i}(t)$ be the estimate of $f(x_{v_1}, \ldots, x_{v_n})$ at node v_i at time t. The natural termination condition would be that $\widehat{f}_{v_i}(t) = f(x_{v_1}, \ldots, x_{v_n})$ for all $v_i \in V$. However, it seems convenient to relax this condition with an approximation factor $\delta \geq 1$, as the following termination predicate

$$Term_{\mathcal{A}}(t) = \left\{ \max_{v_i \in V(t)} \left(\frac{\widehat{f}_{v_i}(t)}{f(x_{v_1}, \ldots, x_{v_n})}, \frac{f(x_{v_1}, \ldots, x_{v_n})}{\widehat{f}_{v_i}(t)} \right) \leq \delta \right\}.$$

In overlay management problems, termination is harder to define, since, as we said, this is an instance of continuous problems. Then, the target is not only to take the overlay network to a state that satisfies some property P (e.g., full connectivity, some degree of expansion), but also to maintain the overlay network in the set of states that satisfy the property. In fact, strictly speaking, an algorithm that provides this service never terminates, since it has to continue running to preserve the property. Reaching the desired property can be seen as a form of *stabilization* of the overlay. We will use the term "termination" for uniformity, and say that the problem has terminated at time t if

$$Term_{\mathcal{A}}(t) = \{P(OV(t))\}.$$

Another issue in overlay management problems is whether the property can in fact be maintained or it can stop being satisfied due to the dynamics of the systems. For instance, many algorithms [1] can still partition the communication system at any time, hence most possibly preventing P from holding.

In the following, we will use $Term_{\mathcal{A}}(t)$ as the predicate to indicate whether a gossip-based algorithm $\mathcal{A}$ has terminated at time t.

2.1.3 Probabilistic Problem Specification

Most gossip-based computations are probabilistic in nature. This may be due to the fact that the algorithms itself is randomized, or that the underlying system only provides probabilistic guarantees. In either case, it very natural to assume that the specification of the problem to be solved could be given in probabilistic terms. These terms usually imply defining some "acceptable" error probability. The error probability can affect the specification in two forms [29]. One is that keeping the complexity fixed (and hopefully small), the answer may have some probability of being incorrect (*Monte Carlo algorithm*). Alternatively, if correctness is to be guaranteed, efficiency is achieved only with some probability (*Las Vegas algorithm*). These two forms of defining probabilistic specifications can be expressed then in the form of a given time complexity C and an error ε, and the condition to be satisfied by both classes of algorithms is that

$$\Pr[Term_{\mathcal{A}}(C)] \geq 1 - \varepsilon. \tag{1}$$

The main difference between the two classes of algorithms has to do with the time the algorithm is run and the condition that defines if the problem has been solved. If a Monte Carlo algorithm is developed, then the algorithm will run for at most C time, and it solves the problem if Equation 1 is satisfied. Observe that in a given execution $Term_{\mathcal{A}}(C)$ may not hold, but that does not prevent the algorithm from stopping. In a Las Vegas algorithm, the execution continues until a time t at which $Term_{\mathcal{A}}(t)$ is satisfied.

2.2 Underlying System Parameters

A gossip-based algorithm is designed to solve the specified problem in a given environment. This environment has to provide with basic elements like communication primitives, but its characteristics may also be hard obstacles to overcome by the algorithm. Here we state some elements of the underlying system that we believe are important to gossip-based algorithms.

2.2.1 Communication Graph

As practically all theoretical models of systems for gossip-based algorithms, we assume that communication between nodes is done via message passing. To model which nodes can directly communicate among each other, we use a communication graph. The *communication graph* of a system model is a graph $G(V, E)$ that consists of the set V of n nodes, each having a unique identifier (i.e., address), interconnected by a set of edges E. Further, a communication channel of bounded capacity, latency and given reliability is associated with each edge. The original gossiping and broadcasting problems assumed a complete underlying communication graph, i.e., a message can be sent between any two nodes. Variants on the communication graph include restrictions on the communication patterns among the nodes, i.e., a node can contact some but not all other nodes. For instance, [14] presents results on the number of calls among n nodes over trees, arbitrary connected graph, and grid graphs.

Note that the communication graph can change over time. If that is the case, we denote by $G(t)$, $V(t)$, and $E(t)$ the graph, active nodes, and available communication channels at time t, respectively. Furthermore, the characteristics of nodes and channels can also change over time. All this has to be included in the model of the underlying system. As a summary, the most important elements of this underlying system are:

Channels The communication channels associated with each edge $e \in E(t)$ have a bounded capacity, latency and given reliability and direction. Consider a wireless sensor network, some devices might be capable of bidirectional transmissions whereas smaller/others not. Another parameter that describes the communication channels is whether the communication is point-to-point or a node can broadcast on all its channels in one single operation. Finally, communication channels can restrict the message length that nodes can use to exchange information.

Network Topology The network, by its nature (e.g., wireless), can restrict the communication patterns by allowing or not different nodes to be connected together via the edge set $E(t)$. Note that the above patterns can change and evolve over time, e.g., mobile peers that have a limited wireless transmission range. We denote the possible communication partners (neighbors) of a node v at time t as $W_v(t) = \{p : (v \in V(t)) \wedge ((v,p) \in E(t))\}$.

Churn Nodes can appear and disappear (become active and inactive) from the system for application (or user-defined) reasons (e.g., join/leave) or for technological reasons (e.g., node crash, failures) at any time. As exposed, $V(t)$ is the set of active nodes at time t.

Overlay It is usually the case that the set of neighbors of every node is very large. Then, for scalability reasons, in this case it is common that only a subset of this set is known and used by v. This is the view that node v locally has of the whole system at a certain time t and is denoted as $view_v(t) \subseteq W_v(t) \subseteq V(t)$. The union of all these views form an overlay of the underlying network topology. The communication between peers (i.e., which node(s) v can pick to communicate with) is thus restricted first by the network (to the set $W_v(t)$) and second by the overlay network (to the set $view_v(t)$).

The underlying system is well-defined when all the underlying system parameters are given, and depending on these parameters, some algorithm parameters are restricted or implicitly fixed.

2.2.2 Overlay Networks and Views

As we said, the overlay network of nodes is created and maintained by maintaining local views of their neighbors at the nodes. The local views that each node maintains (e.g., [1, 7, 11, 16]) are said to be *global* if they contain all the neighbors of the node, and *partial* otherwise. Note that as defined above a view contains only nodes that a given peer can physically contact and is not comparable to *group membership* as defined in [31].

Global Views. Global views have been considered in previous theoretical works in which a gossip-based approach relied on the assumption that each peer locally knows every other peer in the system. Consider, for example, the general structure of a gossip-based protocol discussed in the seminal paper of Demers et al. [5]. The system consists of a set V of n nodes interconnected by a complete graph (clique). In other words, each node has a global knowledge of the system, with a view that contains every other node in the system. The choice of neighbor, be it randomized or deterministic, thus can rely on this global view of the system. In this case, the view maintained by a peer v is equal to the whole network at all times, $view_v(t) = V(t)$, meaning that the nature of the network assumed can only be a complete graph as each peer v can potentially contact any peer in the view, thus network.

Partial View. Providing each node with a global view is unrealistic in a large distributed system for scalability reasons. First the data structure for storing the view should not grow linearly with the system size and second, maintaining such information in the presence of churn incurs considerable communication costs. The resulting problem is a need for protocols that maintain partial views, keeping given desired properties of the overlay network (e.g., connectivity [1, 7]). Interestingly, the problem of overlay maintenance can itself be solved by gossip-based algorithms (e.g., [1, 7, 11, 16, 37]). Very often the overlay maintenance is achieved with a *Peer Sampling Service* (named after [16]).

2.2.3 Peer Sampling Service

A peer sampling service [1, 7, 11, 16, 37] provides nodes with samples of the set $V(t)$, which have some probabilistic guarantees (typically, is a uniformly chosen random sample). The sampling has to consider the churn rate experienced by the system, trying to prevent including inactive nodes in the samples. In the above cited work, the peer sampling service always assumed an underlying complete communicating graph, and used the sample as the node's view, such that $view_v(t) \subseteq W_v(t) = V(t)$. It would be of great interest to define generic peer sampling algorithms that (1) take into account the network restrictions, such that $view_v(t) \subseteq W_v(t) \subseteq V(t)$ where $W_v(t)$ can arbitrarily change over time, (2) try to reduce the load on the network, and (3) finally keep given desired properties of the overlay network.

2.3 Gossip-Based Algorithms

In this section we present the structure of a generic gossip-based algorithm and identify its most significant parameters.

2.3.1 Structure of Gossip-Based Algorithms

Gossip-based algorithms have a very simple and regular structure. The code of the gossip-based algorithm is executed by each node in rounds. Every round r, a node performs (1) a *communication phase*, followed by (2) a *processing phase*.

Communication Phase. In the communication phase of a round, a network node v chooses a subset of communication partners (called *neighbors*) from its local view, and exchanges with them information it holds. The message sent by the node is synthesized from the current state of the node, and possibly includes information from several previous exchanges.

Processing Phase. After the communication phase, a network node applies a state transition function to its current state to obtain the new state. The transition depends on the current state and the information obtained form the set of neighbors that have been contacted. In the case of information spread or broadcast, the state transition function defines (1) what information should be delivered (i.e., if the node received new information) and (2) what information has to be gossiped in the next round(s) (i.e., how long an information has to be gossiped, named *contagion period*).

We illustrate a common structure of a generic gossip-based algorithm in Figure 1. The communication phase begins in line 2, as the neighbors set is filled by a SELECT method. This method implements the *communication strategy* of the algorithm, defined by the underlying system parameters and adjusted by the *algorithm parameters.* For each of the chosen communication partners, a communication channel is opened and data can be exchanged (line 4). The gossip message is exchanged depending on the *transmission model* chosen for the algorithm (lines 6, 9, 16 and 20), and after the information is received (lines 9 and 16), the communication phase is finished, and the processing phase starts. The UPDATE method implements the state transition function that will effectively solve the problem, delivers the information and chooses what information to gossip in the following round(s) (if applicable) and finally does some maintenance tasks (e.g., *buffer management*).

```
 1: upon TIMER($t$ time units) at node $v_i$ do
 2:    $neighbors$ ← SELECT $F$ communication partners from $view_{v_i}$
 3:    for all $p \in$ $neighbors$ do
 4:       COMMUNICATEWITH($p$)
 5:       if $push$ then
 6:          SEND gossip message to $p$
 7:       end if
 8:       if $pull$ then
 9:          RECEIVE gossip message from p
10:          UPDATE local state
11:       end if
12:    end for
13: end upon

14: upon COMMUNICATEWITH() from $v_j$ at processor node $v_i$ do
15:    if $push$ then
16:       RECEIVE gossip message from $v_j$
17:       UPDATE local state
18:    end if
19:    if $pull$ then
20:       SEND gossip message to $v_j$
21:    end if
22: end upon
```

Figure 1: Generic gossip-based algorithm pseudocode. *The booleans push and pull are true in case of a pushpull transmission model.*

2.3.2 Algorithm Parameters

The above basic pseudocode of a gossip-based algorithm has to be instantiated depending on the problem that has to be solved and the underlying available system. Among others, this instantiation depends on the following basic parameters.

Transmission Model. As can be observed, in Figure 1, there are two flags, *push* and *pull*, that strongly characterize the behavior of the algorithm. They determine if the communication of the node with the neighbors only transfers information from the node to the neighbor, only transfers information from the neighbor to the node, or transfers information in both directions. The first two cases are one-way transmission. In the first, we say that information is *pushed* from the node to the neighbor, while in the second we say that the information is *pulled* by the node from the neighbor. In the general case of two-way transmission, where the two nodes exchange their information, the transmission model is named *pushpull.*

Communication Strategy and Fanout. The communication strategy defines how a node v chooses the subset of communication partners, i.e., its *neighbors*, for the current round. The first decision the algorithm designer has to make is the size of the neighbors' set, F, known as the *fanout.* The set of neighbors is chosen from the $view_v(t_r)$ (t_r is the time at which round r is executed), thus $neighbors \subseteq view_v(t_r) \subseteq W_v(t_r) \subseteq V(t_r)$. The way a set of neighbors is chosen can be *deterministic* or *random.* Following the communication strategies defined, it is sometimes the underlying system that dictates or influences the choice of neighbor(s) with whom communicate by reflecting the nature of the underlying network (e.g., network-driven communication strategy [20, 26]) or to reflect a given arbitrary topology (e.g., application-driven topology [15]).

Buffer Management. Inherently to all gossip-based algorithm, duplication of messages can happen. Furthermore, it is common that every message carries several units of information, typically named *events.* Hence, it is highly possible that a node receives information (events) that it already knew, which is one of the reasons gossip-based algorithms are considered fault-tolerant. With random communication, there is a fair chance that the same neighbor is chosen more than once. Moreover, even without repeating partners, it is possible that two nodes p and q, which both communicated with r in the past, can now exchange r's information.

For scalability reasons (e.g., overall number of messages exchanged, size of these messages, etc.) an algorithm usually specifies that a node should not forward all events forever and decide to stop gossiping this or that particular event (i.e., by removing it from the gossip message) at some point. This *buffer management* problem is exposed in [8] and solutions are proposed in, e.g., [7, 32].

Message Size. Within the restrictions imposed by the underlying system, the algorithm has to decide how to forward the events that it has in its buffer. It may decide that it will forward all events forever, which means that unless events are purged from the memory the message size will grow arbitrarily large. Another option is to decide that only a fixed number of events will be forwarded in each message, which implies that messages will have a fixed size, but open the question of how to select the set of events to transmit (e.g., age-based purging [7]). Finally, it can decide to forward each event a maximal number of times, which implies that the message size depends on the number of events to gossip in the following round(s). In epidemic terminology, each different event represents an infectious disease and sending or receiving messages translates to infecting and being infected by other nodes. Once a node gets infected by a disease, it can basically (1) decide it can only be infectious by a fixed number of diseases and thus instantly heal other diseases (i.e., having a maximal message size), (2) be infectious forever (i.e., forwarding each event an infinite number of times), or (3) be infectious during a given time, named *contagion period* (i.e., forwarding the same event a given number of times since reception of it), a special case is named *infect-and-die* when the contagion period is 1 for a given disease.

History Buffer Size. A related issue is to manage the history of delivered events. The history of delivered events is maintained on each node in order to decide if a received information is new (i.e., has to be delivered in case of information spread), or has already been received in the past (i.e., the node received a duplicate). The management of the delivered buffer must be scalable as more and more events are received (and delivered) over time. Modeling and analysis of the history buffer can be found in [23].

3. COMPLEXITY

A gossip-based algorithm is expected to solve the intended problem in an efficient manner. However, to decide if an algorithm is efficient, we need some measurement units to compare algorithms with each other. The performance of gossip-based algorithms is usually measured with the following criteria, with which smaller values imply more efficient algorithms.

Time complexity. The time complexity of an algorithm $\mathcal{A}$ is the time it takes to solve the problem. Using the notation introduced in Section 2 for the termination event, we can define this complexity as

$$T_{\mathcal{A}} = \inf\{t : \mathit{Term}_{\mathcal{A}}(t)\}.$$

That is, the time complexity of the algorithm is the time it takes to satisfy the termination condition. This definition of time complexity is appropriate for deterministic behaviors and Las Vegas probabilistic algorithms. However, randomized gossip-based algorithms are usually considered Monte Carlo algorithms. That is, after a fixed time of executing the gossip-based algorithm, there is some probability of having reached termination. In this set up, it seems more appropriate to define an acceptable error probability $\varepsilon \geq 0$, and hence define the time complexity of algorithm $\mathcal{A}$ as

$$T_{\mathcal{A}}(\varepsilon) = \inf\{t : \Pr[\mathit{Term}_{\mathcal{A}}(t)] \geq 1 - \varepsilon\}.$$

As described in Section 2, gossip-based algorithms work in rounds. If these rounds are (roughly) synchronous, in the sense that all nodes complete the same number of rounds in the same time, the above definition can be translated from time to rounds. This is frequently done, and very often the time complexity analysis is done in terms of rounds instead of time. In this case, if $\mathit{Term}_{\mathcal{A}}(r)$ denotes the event that algorithm $\mathcal{A}$ has terminated at the end of round r, the time complexity measured in rounds of $\mathcal{A}$ is defined as

$$T^r_{\mathcal{A}}(\varepsilon) = \inf\{r : \Pr[\mathit{Term}_{\mathcal{A}}(r)] \geq 1 - \varepsilon\}.$$

Observe that for overlay management algorithms (and any other continuous problem), since they never really terminate, the time complexity in fact gives the time it takes for the algorithm to reach a "stable" state in which the desired property is satisfied. This is clearly an interesting performance parameter that efficient algorithms should try to minimize.

Connectivity complexity. The connectivity complexity is the total number of communication channels established between nodes until termination. Given the time complexity in rounds of a gossip-based algorithm, it is usually fairly easy to bound the connectivity complexity. Consider for example, an arbitrary gossip-based algorithm $\mathcal{A}$ that runs over network $G(V, E)$ of n nodes, which exhibits time complexity, $T^r_{\mathcal{A}}$. Assume that, in every round, each node establishes at most a constant number c of connections. Thus, $\mathcal{A}$'s connectivity complexity $C_{\mathcal{A}}$ is bounded as follows

$$C_{\mathcal{A}} \leq c \times n \times T^r_{\mathcal{A}}.$$

Observe that overlay management algorithms (continuous problems) never really terminate. For them, on top of the connectivity complexity until "termination", an interesting parameter is the *connectivity rate*, which is the number of connections or communication channels established per unit of time once the stable state has been reached. If the algorithm works in synchronous rounds the connectivity rate can be defined in terms of connections per round. With the above assumptions, if algorithm $\mathcal{A}$ is an overlay management algorithm, its connectivity rate $CR_{\mathcal{A}}$ would be bounded as

$$CR_{\mathcal{A}} \leq c \times n.$$

Message complexity. The message complexity of an algorithm is the total amount of data transferred over the network channels throughout an execution until termination. A bound on this parameter can be easily computed from the connectivity complexity and the maximum message size. Let m be an upper bound on the size of the messages exchanged by algorithm $\mathcal{A}$, in bits. Assuming that all connections exchange messages in both directions (pushpull case), it is fairly easy to bound the message complexity $M_{\mathcal{A}}$ of algorithm $\mathcal{A}$ as

$$M_{\mathcal{A}} \leq 2 \times C_{\mathcal{A}} \times m.$$

For overlay management algorithms, it is possible to define the *message rate* as the amount of data exchanged per unit of time once the stable state has been reached. Trivially, this value can be upper bounded by $2 \times CR_{\mathcal{A}} \times m$.

Space complexity. The space complexity of an algorithm is the amount of memory devoted to the algorithm in each node. This memory has been mostly used for storing overlay data (i.e., view), message buffering and history management. The measurement of these sets should be given as a function of the network size (n) for the overlay data and as a function of the message complexity for message and history data.

4. CONCLUSION AND FUTURE WORK

In this paper we have presented a generic framework for theoretical analysis of gossip-based algorithms. In this framework we have identified the key elements that are involved in the evaluation of an algorithm, namely the problem it solves, the environment on which it runs, and the goodness measures that can be used to evaluate it. Hopefully, this work will open the door to many future lines of (theoretical) research. We point out some issues that we believe should be looked at further in the rest of this section.

Layering. In our framework we have identified three layers in a gossip-based computational environment: the service, the system, and the algorithm itself. We have placed the overlay as part of the underlying system, and mentioned that it can be managed as a service that can be provided with a gossip-based algorithm as well. This leads to a new layering. The issue we want to raise is whether any gossip-based computation can in fact be cleanly divided into these layers. It would be nice to prove whether this is always possible or, on the contrary, there are cases in which a strong coupling between layers is required. Additionally, in the first case, it would be of interest to determine whether this layering has some impact on the performance of the algorithm.

Robustness to Sampling. In our framework we placed the peer sampling service as part of the environment. In theoretical analysis it is often assumed that this service has some "nice" properties, like uniform randomness. However, as mentioned above, this service itself could be provided by a gossip-based algorithm and the service may not be as "nice" as expected. It seems of interest to evaluate the robustness of the guarantees of gossip-based algorithms to imperfect peer sampling services. For instance, if an algorithm works correctly for uniform random sampling, how robust it is if the sampling is not exactly uniform? Which properties still hold?

Modeling Dynamics (Fault Tolerance). We have included in the framework the fact that current networks are highly dynamic. Gossip-based algorithms have been shown empirically to provide the robustness required to perform useful work in the presence of this constant change. Thus, we believe that an important direction of theoretical study is the analysis of the behavior of gossip-based algorithms in the presence of network dynamics. For that, a model of the dynamic behavior of nodes and links has to be defined. Some concepts from classical distributed systems can be used, like the classification of node failures. However, new parameters may need to be considered, like changes in the network topology or the link bandwidths. An inspiring previous work is [18], in which Karp et al. investigate the robustness of their *median counter* gossip algorithm against failures. Their model, may be adopted as general settings in order to qualify the robustness of gossip style algorithms.

Sequences of Problems. In our framework we have identified three main classes of problem typically solved with gossip-based algorithms. To our knowledge still largely unexplored is a new class of problems in which we have a continuous arrival of instances of basic problems. An instance of this class could be a problem in which broadcasting problems are continuously arriving, and all the broadcasts have to be efficiently completed. This new class requires new parameters and models. For instance, a parameter of the problem which is of great interest is the arrival rate of new instances, and a performance metric is at which arrival rate the gossip-algorithm collapses and is not able to complete all the problem instances.

Continuous Problems. We have mentioned that the overlay management problem has the interesting property that it really never terminates, since it is a continuously available service. It would be interesting to identify other problems that fall in this category, and exhaustively identify parameters of interest for their modeling and analysis.

Gossip Symmetry. Most of the gossip algorithms in the literature consider that each peer executes the same implementation of the algorithm with the algorithm parameters (e.g., every peer has the same fanout). It would be interesting to see the impact of arbitrary distributions on some given system settings (e.g., fanout, contagion period) on the different complexity measurements that we proposed. The closest related works to this open research direction are [11] and [21] where the authors expose, following [2], that the probability of having an atomic broadcast in the infect-and-die model is only dependent on the mean fanout and not on the exact distribution of it, but no results that we know of have yet shown that the distribution has no impact on any complexity measures.

Fanin Limitation. In a highly scalable gossip-based algorithm, the memory requirement (with respect to space complexity) and communication requirement (with respect to message and connectivity complexity) should ideally not change with the network size $|V| = n$. However, few gossip-based algorithms limit the number of incoming connections. Consider the general structure of the *anti-entropy* protocol of Demers et al. [5]. In each cycle every node chooses uniformly at random a single communication partner from all the network nodes and pulls information from it. In [22] Koldehofe provides a method for analyzing such protocols, by representing the propagation of information as a balls-and-bins random process [29]. Formally, suppose we have n bins and we uniformly at random throw n balls into them. Thus, the maximum number of incoming connections a node may experience during a single execution cycle bears resemblance to the maximum number of balls in any bin. This maximum load is known to be $O(\frac{\log n}{\log \log n})$ with high probability [30], which is a function of the size of the network n. In essence, we require a scalable gossip-based algorithm to employ communication strategies in which both outgoing and incoming connections to and from each node are bounded by a constant number.

Acknowledgments

The authors would like to thank Michel Raynal and Vivek Rai for useful discussions and contributions to the contents of this paper. Antonio Fernández has been partially funded by the Spanish Ministry of Education with grant TIN2005-09198-C02-01 and the Comunidad de Madrid with grant S-0505/TIC/000398. Maxime Monod has been partially funded by the Swiss National Science Foundation with grant 200021-113825.

5. REFERENCES

[1] A. Allavena, A. Demers, and J. E. Hopcroft. Correctness of a gossip based membership protocol. In *PODC '05: Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, pages 292–301, 2005.

[2] F. Ball and A. Barbour. Poisson approximation for some epidemic models. *J. Applied Probability*, 27:479–490, 1990.

[3] A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder. Grapevine, an exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, 1982.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: Design, analysis and applications. In *INFOCOM '05: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1653–1654, 2005.

[5] A. J. Demers, D. H. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. E. Sturgis, D. C. Swinehart, and D. B. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[6] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *IPSN '06: Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, pages 69–76, 2006.

[7] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[8] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, 2004.

[9] P. T. Eugster, R. Guerraoui, and P. Kouznetsov. Δ-reliable broadcast: A probabilistic measure of broadcast reliability. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 636–643, 2004.

[10] M. Flammini and S. Pérennes. Lower bounds on systolic gossip. *Inf. Comput.*, 196(2):71–94, 2005.

[11] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.

[12] R. Guerraoui, S. B. Handurukande, K. Huguenin, A.-M. Kermarrec, F. Le Fessant, and E. Rivière. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In *P2P '06: Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, pages 12–22, 2006.

[13] Z. J. Haas, J. Y. Halpern, and L. Li. Gossip-based ad hoc routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, 2006.

[14] S. Hedetniemi, S. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(1):319–349, 1988.

[15] M. Jelasity and Ö. Babaoglu. T-man: Gossip-based overlay topology management. In *ESOA'05: Proceedings of the 3rd International Workshop on Engineering Self-Organising Systems*, pages 1–15, 2005.

[16] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 79–98, 2004.

[17] B. H. H. Juurlink, J. F. Sibeyn, and P. S. Rao. Gossiping on meshes and tori. *IEEE Trans. Parallel Distrib. Syst.*, 9(6):513–525, 1998.

[18] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[19] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 482, 2003.

[20] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. *J. ACM*, 51(6):943–967, 2004.

[21] A.-M. Kermarrec, L. Massoulié;, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, 2003.

[22] B. Koldehofe. Simple gossiping with balls and bins. In *In Proceedings of the 6th International Conference on Principles of Distributed Systems (OPODIS'02).*, pages 109–118, 2002.

[23] B. Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *SRDS '03: Proceedings of the 22ns Symposium on Reliable Distributed Systems*, pages 76–85, 2003.

[24] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* Morgan Kaufmann, San Mateo, 1992.

[25] K. Lidl, J. Osborne, and J. Malcome. Drinking from the firehose: Multicast usenet news. In *Proceedings of the Usenix Winter Conference*, pages 33–45, 1994.

[26] M.-J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *EDCC '99: Proceedings of*

the 3rd European Dependable Computing Conference, pages 364–379, 1999.

[27] U. Meyer and J. F. Sibeyn. Oblivious gossiping on tori. *J. Algorithms*, 42(1):1–19, 2002.

[28] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *PODC '06: Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, pages 113–122, 2006.

[29] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, New York, NY, USA, 1995.

[30] M. Raab and A. Steger. "balls into bins" - a simple and tight analysis. In *RANDOM '98: Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170, 1998.

[31] A. Ricciardi and K. P. Birman. Using process groups to implement failure detection in asynchronous environments. In *PODC '91: Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 341–353, 1991.

[32] L. Rodrigues, S. Handurukande, J. Pereira, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *DSN '03: Proceedings of the International Conference on Dependable Systems and Networks*, pages 47–56, 2003.

[33] J. F. Sibeyn. Faster gossiping on butterfly networks. *Theor. Comput. Sci.*, 331(1):53–72, 2005.

[34] J. F. Sibeyn and M. Soch. Optimal gossiping on cccs of even dimension. *Parallel Processing Letters*, 13(1):35–42, 2003.

[35] R. van Renesse. Scalable and secure resource location. In *HICSS '00: Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences*, pages 10–19, 2000.

[36] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middleware '98: Proceedings of 1st IFIP International Conference on Distributed Systems Platform and Open Distributed Processing*, pages 55–70, 1998.

[37] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.