Workshops in Computing Series, Springer-Verlag (1995) 249–263.

Keeping Track of the Latest Gossip in Message-Passing Systems

Madhavan Mukund* K. Narayan Kumar† Milind Sohoni‡

Abstract

Consider a distributed system in which processes exchange information by passing messages. The *gossip problem* is the following: Whenever a process q receives a message from another process p, q must be able to decide which of p and q has more recent information about r, for every other process r in the system. With this data, q is in a position to update its knowledge about the global state of the system.

We propose a solution where each message between processes carries information about the current state of knowledge of the sender. This information is *uniformly* bounded if we make reasonable assumptions about the number of undelivered messages present at any time in the system. This means that the overhead of maintaining the latest gossip is a constant, independent of the length of the underlying computation.

Introduction

We tackle a natural problem from distributed computing, involving time-stamps. Let $\mathcal{P}$ be a set of computing agents or processes which exchange information by passing messages. The *gossip problem* is the following: Whenever a process q receives a message from another process p, q must be able to decide which of p and q has more recent information about r, for every other process r.

By keeping track of the latest gossip about other agents, each process can consistently update its knowledge about the global state of the system whenever it receives some new information from another process. Computing global information about the system from local information is, of course, a central issue in distributed computing. So, a solution to the gossip problem would be useful in a wide variety of applications involving distributed systems.

The gossip problem has been investigated in [5, 9] for systems where processes synchronize periodically and exchange information. We extend the solution proposed in [5, 9] to a general message-passing model, where processes communicate by sending messages on point-to-point channels. We assume that communication is guaranteed—all messages in the system are eventually delivered. However, we permit indefinite delays in transit. Also, messages need not be received in the order in which they were sent.

In our solution to the gossip problem, processes exchange only a bounded amount of gossip information with each message they send. At the heart of our

*School of Mathematics, SPIC Science Foundation, 92 G.N. Chetty Road, T. Nagar, Madras 600 017, India. E-mail: madhavan@ssf.ernet.in

†Computer Science Group, Tata Institute of Fundamental Research, Homi Bhabha Road, Colaba, Bombay 400 005, India. E-mail: kumar@tcs.tifr.res.in

‡Department of Computer Science and Engineering, Indian Institute of Technology, Bombay 400 076, India, E-mail: sohoni@cse.iitb.ernet.in

solution is a protocol for time-stamping messages in the system using a finite set of labels.

Time-stamping is a well-established technique for ordering events in a distributed setting [6, 7]. Time-stamping protocols which use only a bounded set of labels to tag events have attracted a fair amount of attention in recent years. Protocols have been exhibited for systems in which processes communicate via a shared memory [3, 4], as also for systems where processes synchronize periodically and exchange information [1, 2, 9]. However, no such protocol seems to exist for message-passing systems.

It is not difficult to see that in the completely general message-passing model described above, bounded time-stamping is not possible. In order to make the problem tractable, we have to restrict the model by placing a bound on the number of unacknowledged messages that can be present in the system at any time. We believe that this restriction is a natural one and that all "reasonable" distributed algorithms for message-passing systems would actually conform to the paradigm we propose.

An important feature of our solution to the gossip problem is that it does not introduce any additional messages—it just adds some additional data to each message of the underlying computation. This additional data is guaranteed to be uniformly bounded. So, given any distributed algorithm which conforms to the restricted model we work with, we can enhance the algorithm to also keep track of the latest gossip with only a constant overhead in message complexity.

The paper is organized as follows. In the next section we introduce our model of computation and formulate the gossip problem in terms of a natural partial order on the events in the system. Section 2 describes ideals, which capture the notion of a partial view of a distributed computation. Sections 3 and 4 describe a protocol to solve the gossip problem for *fifo* computations, where messages are delivered in the order they were sent. Each process maintains what we call *primary information* about the computation, using potentially unbounded labels to distinguish messages in the system. In Section 5 we show how to convert this protocol to one which uses bounded time-stamps, thereby establishing a solution to the gossip problem in the fifo case. In the final section, we give a quick sketch of how to extend this solution to the case where messages are not necessarily delivered in the order in which they were sent.

1 The Model

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ be a set of processes which communicate with each other through messages. We assume that messages are never lost—that is, the communication medium is reliable. However, there may be an arbitrary delay between the sending of a message and its receipt. Further, messages need not be received in the order in which they were sent.

We assume that communication is point-to-point. So, each message is addressed to a specific process and is not seen by any of the other processes in the system. Thus, each transmission of a message from a process p to a process q consists of two distinct actions; the action $s_{p \to q}$ corresponds to the sending of the message from p to q and the action $r_{q \leftarrow p}$ corresponds to its receipt by q.

We can regard a computation of the system as a word over the alphabet $\mathcal{C} = \mathcal{C}_S \cup \mathcal{C}_R$ where $\mathcal{C}_S = \{s_{p \to q} \mid p, q \in \mathcal{P}\}$ is the set of *send actions* and

$\mathcal{C}_R = \{ r_{p \leftarrow q} \mid p, q \in \mathcal{P} \}$ is the set of *receive actions*. Each action in $\mathcal{C}$ is "executed" by a single process. So, we can also partition $\mathcal{C}$ across processes—for each process p, $\mathcal{C}_p = \{ s_{p \rightarrow q} \mid q \in \mathcal{P} \} \cup \{ r_{p \leftarrow q} \mid q \in \mathcal{P} \}$ is the set of *p-actions* that p participates in directly.

We shall regard a word $u \in \mathcal{C}^*$ of length m as a function $u : [1..m] \rightarrow \mathcal{C}$, where $[1..m]$ denotes the set $\{1, 2, \ldots, m\}$ if $m \geq 1$ and is $\emptyset$ if $m = 0$. For $u \in \mathcal{C}^*$ and $c \in \mathcal{C}$, $\#_c(u)$ denotes the number of occurrences of c in u. We can extend this to subsets $X \subseteq \mathcal{C}$: $\#_X(u) = \sum_{c \in X} \#_c(u)$.

Not every word corresponds to a valid computation—for instance, we must insist that messages are received only after they are sent. In addition, since messages need not be received in the order they were sent, to completely specify a computation we need to match each receive event to the corresponding send event. With this in mind, we define computations as follows:

Computations A *computation* over $\mathcal{C}$ is a pair (u, ϕ) where $u : [1..m] \rightarrow \mathcal{C}$ is a word and $\phi : [1..m] \rightharpoonup [1..m]$ is a partial function such that:

(i) The domain of ϕ, $dom(\phi)$ is $\{ i \mid u(i) \in \mathcal{C}_R \}$.

(ii) ϕ is injective over $dom(\phi)$—for each $i, j \in dom(\phi)$, $i \neq j \Rightarrow \phi(i) \neq \phi(j)$.

(iii) For each $i \in dom(\phi)$, $\phi(i) < i$.

(iv) If $u(i) = r_{q \leftarrow p}$ then $u(\phi(i)) = s_{p \rightarrow q}$.

If $\phi(i) = j$, then $u(i)$ is a receive action whose corresponding send action is $u(j)$. By condition (ii), we may also refer to i unambiguously as $\phi^{-1}(j)$.

EXAMPLE: *Let $\mathcal{P} = \{p, q\}$ and let u be the string $s_{p \rightarrow q} s_{p \rightarrow q} s_{p \rightarrow q} r_{q \leftarrow p} r_{q \leftarrow p}$ and ϕ be the function where $\phi(4) = 3$ and $\phi(5) = 1$. So, in this computation, the message sent from p to q at $u(1)$ is overtaken by the message sent at $u(3)$. Moreover, the message sent at $u(2)$ has not reached q at all.*

Events and causality The word u imposes a total, temporal order on the actions observed during a computation (u, ϕ). However, in order to analyze the flow of information between processes, we need a more accurate description of the cause and effect relationship between the different actions in u.

Let (u, ϕ) be a computation, where $u : [1..m] \rightarrow \mathcal{C}$. We associate with (u, ϕ) a set of events $\mathcal{E}_u = \{ (i, u(i)) \mid i \in [1..m] \}$.

Let $e = (i, u(i))$ be an event in $\mathcal{E}_u$. When there is no ambiguity, we shall use e to denote both i and $u(i)$. For instance, $e \in \mathcal{C}_p$ denotes that $u(i) \in \mathcal{C}_p$—in other words, e is a *p-event*. Similarly, if we say $f = \phi(e)$ we mean that $f = (j, u(j))$ is an event such that $\phi(i) = j$. We shall also use $\mathcal{E}_u$ and u interchangeably in expressions such as $\#_c(\mathcal{E}_u)$, which denotes $\#_c(u)$.

As we mentioned earlier, u imposes a total, temporal order on the events in $\mathcal{E}_u$. Let $e, f \in \mathcal{E}_u$. Then $e < f$ provided $e = (i, u(i))$, $f = (j, u(j))$ and $i < j$. As usual $e \leq f$ if $e < f$ or $e = f$.

Messages introduce causality across processes. For each pair $(p, q) \in \mathcal{P} \times \mathcal{P}$ such that $p \neq q$, define $\lhd_{pq}$ to be the ordering

$$e \lhd_{pq} f \stackrel{\triangle}{=} e \in \mathcal{C}_p, f \in \mathcal{C}_q \text{ and } \phi(f) = e.$$

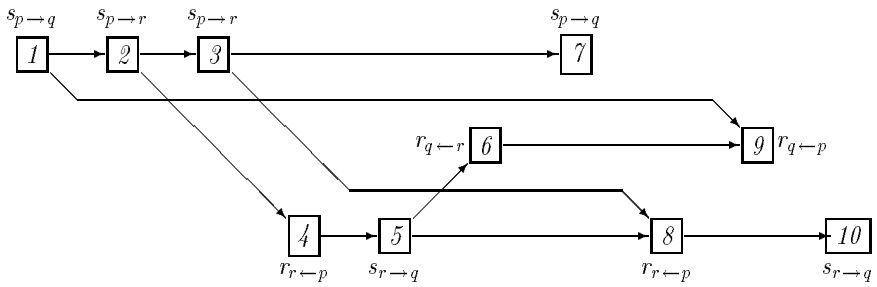

Figure 1: An example

In addition, each process p orders the events it participates in. Define $\lhd_{pp}$ to be the strict ordering

$$e \lhd_{pp} f \stackrel{\triangle}{=} e < f,\ e \in \mathcal{C}_p,\ f \in \mathcal{C}_p \text{ and for all } e < g < f,\ g \notin \mathcal{C}_p.$$

The set of all p-events in $\mathcal{E}_u$ is totally ordered by $\lhd_{pp}^*$, the reflexive, transitive closure of $\lhd_{pp}$.

Define $e \sqsubset f$ if for some $p, q \in \mathcal{P}$ (where p and q need not be distinct), $e \lhd_{pq} f$ and $e \sqsubseteq f$ if $e = f$ or $e \sqsubset f$. Let $\sqsubseteq^*$ denote the transitive closure of $\sqsubseteq$. If $e \sqsubseteq^* f$ then we say that e is *below* f. The partial order $\sqsubseteq^*$ records the information we require about causality and independence between events in $\mathcal{E}_u$.

Let $e \in \mathcal{E}_u$ be a p-event. The set of events below e is $e{\downarrow} = \{f \mid f \sqsubseteq^* e\}$. These represent the only actions which are known to p when e occurs.

Latest information Consider a computation (u, ϕ) and its associated set of events $\mathcal{E}_u$. The $\sqsubseteq^*$-maximum p-event in $\mathcal{E}_u$ is denoted $max_p(\mathcal{E}_u)$—this is the last event in $\mathcal{E}_u$ in which p has taken part. This quantity is well defined whenever $\#_{\mathcal{C}_p}(\mathcal{E}_u) > 0$, since all p-events in $\mathcal{E}_u$ are totally ordered by $\sqsubseteq^*$. (Recall that $\mathcal{C}_p \subseteq \mathcal{C}$ is the set of p-actions, so $\#_{\mathcal{C}_p}(\mathcal{E}_u)$ denotes, by convention, the number of p-actions mentioned in the string u.)

Let $p, q \in \mathcal{P}$. If $\#_{\mathcal{C}_p}(\mathcal{E}_u) > 0$, the latest information p has about q in $\mathcal{E}_u$ corresponds to the $\sqsubseteq^*$-maximum q-event in the set $max_p(\mathcal{E}_u){\downarrow}$, provided the set of q-events below $max_p(\mathcal{E}_u)$ is not empty. We denote this event by $latest_{p\leftarrow q}(\mathcal{E}_u)$. (If there are no q-events in $max_p(\mathcal{E}_u){\downarrow}$, then $latest_{p\leftarrow q}(\mathcal{E}_u)$ is undefined.)

EXAMPLE: *Let $\mathcal{P} = \{p, q, r\}$. Consider the computation (u, ϕ), where $u = s_{p\to q}\, s_{p\to r}\, s_{p\to r}\, r_{r\leftarrow p}\, s_{r\to q}\, r_{q\leftarrow r}\, s_{p\to q}\, r_{r\leftarrow p}\, r_{q\leftarrow p}\, s_{r\to q}$, $\phi(4) = 2$, $\phi(6) = 5$, $\phi(8) = 3$ and $\phi(9) = 1$. Figure 1 is a picture of $(\mathcal{E}_u, \sqsubseteq^*)$. The arrows in the figure correspond to the basic relations $\lhd_{pq}$, which generate $\sqsubseteq^*$.*

In this computation, $max_q(\mathcal{E}_u) = (9, r_{q\leftarrow p})$. Though at $max_q(\mathcal{E}_u)$, process q hears from process p, $latest_{q\leftarrow p}(\mathcal{E}_u)$ does not correpond to $\phi(max_q(\mathcal{E}_u))$. Instead, $latest_{q\leftarrow p}(\mathcal{E}_u) = (2, s_{p\to r})$—process q hears this information indirectly, via process r.

The gossip problem

Let p, q and r be processes such that $latest_{p \leftarrow r}(\mathcal{E}_u)$ and $latest_{q \leftarrow r}(\mathcal{E}_u)$ are both defined. Since both of these are r-events, they must be ordered by $\lhd_{rr}^*$. So, the latest information that p and q have about r will always be comparable.

The gossip problem is the following.

> *Whenever a process q receives a message from another process p, q must be able to decide which of p and q has heard more recently from r, for every other process r in the system.*

One way to resolve this problem is as follows. As the computation progresses, each action is assigned a label by the process involved in that action. These labels allow processes to refer to events in an unambiguous manner. Each process then maintains the labels corresponding to its latest information. These labels are passed on with each communication in such a way that the process receiving the message can consistently update its own latest information.

The labels which are assigned to events during a computation are essentially *time-stamps*. A trivial solution to the time-stamping problem is for each process to maintain a local counter and assign strictly increasing counter values to the actions it executes. Then, two processes p and q can compare their latest information about r by checking which of their "latest" r labels is larger.

This scheme has the following drawback: As the computation progresses, the time-stamps assigned to events grow without bound. As a result, processes need to send longer and longer messages to transfer the labels corresponding to their latest information.

We seek a solution to the gossip problem where message lengths are bounded. This will ensure that the overhead of maintaining gossip information remains a constant, regardless of the length of the underlying computation.

To achieve this, we need to devise a scheme for labelling events using a bounded set of time-stamps. This means that, eventually, the same time-stamp will be assigned to more than one event. We need to ensure that time-stamps are reused in such a way that the update of latest information is not affected.

In principle, this should be possible. Let $\mathcal{E}_u$ be the events corresponding to the computation (u, ϕ) and let the number of processes in the system be N. Regardless of the number of events in $\mathcal{E}_u$, at most N^2 of them are relevant for solving the gossip problem—we only need to be able to compare the labels of events of the form $latest_{p \leftarrow q}(\mathcal{E}_u)$ for each pair $p, q \in \mathcal{P}$. So, in effect, at most N^2 of the events in $\mathcal{E}_u$ constitute "current" gossip. Moreover, once an event becomes "obsolete" its time-stamp can be safely reused—an "obsolete" event can never become "current" at a later stage in the computation.

However, in the completely general model we have considered so far, it is impossible to achieve our goal. Since messages can be delayed indefinitely, a process p may send unboundedly many messages to q without knowing whether any or all of them have reached. Until p receives some confirmation from q that a particular message has reached, that message's time-stamp cannot be reused. Thus, p will potentially need to use an unbounded number of time-stamps to label its messages to q. (Notice that this problem arises even if messages are delivered in the order in which they were sent—the main source of difficulty is the fact that there is no bound on the delay in delivering a particular message.)

B-bounded computations To overcome this problem, we need to restrict the class of computations we permit. Intuitively, we must bound the number of unacknowledged messages between any pair of processes. More formally, we say that (u, ϕ) is a *B-bounded computation* provided the following holds:

$$\text{For each event } e = (i, s_{p \to q}) \text{ in } \mathcal{E}_u,\ \#_{s_{p \to q}}(e\downarrow) - \#_{r_{q \leftarrow p}}(e\downarrow) \leq B.$$

In other words, p can send a fresh message to q only if, as far as it knows, the number of messages which it has already sent to q and which are yet to be delivered is less than B. Notice that p need not get direct acknowledgments from q. For instance, q could tell r that it has received a particular message m from p and r, in turn, could pass on this information to p.

Even for B-bounded computations, it is not immediate that the gossip problem has a solution. Suppose process p sends process q a message m. Though p is guaranteed to receive an acknowledgment for this message by the time it sends its next B messages to q, it cannot naïvely reuse m's time-stamp once m is acknowledged. In between, q may have passed on the information in m to another process r, in which case the message m would still constitute "current" gossip for r. So, p has to have some means of recording which of its time-stamps are "in use" in the system at any given time.

Fifo computations Though we eventually present a solution to the gossip problem for all B-bounded computations, it will be convenient to first look at the problem in a restricted setting, where messages are delivered in the order in which they were sent. Formally, we say that a computation (u, ϕ) is *fifo* if the following holds: For every pair of processes p and q and for any two receive events e and f of the form $r_{p \leftarrow q}$, $e \sqsubset^+ f$ implies $\phi(e) \sqsubset^+ \phi(f)$, where $\sqsubset^+$ denotes the transitive closure of the relation $\sqsubset$.

Though messages between processes may not overtake each other in a fifo computation, gossip information along one route may overtake gossip information on another route. For instance, suppose p sends a message m to q and then sends a message m' to r, following which r sends a message m'' to q which reaches q before m does. Then, by the time m reaches q, the information that m contains about the state of p is obsolete. So, even in a fifo computation, each process has to have a way of resolving whether the message it has just received has already been superseded by earlier, indirect information.

For the next four sections, we assume that every computation we deal with is B-bounded and fifo.

2 Ideals

Let us fix a computation (u, ϕ), where $u : [1..m] \to \mathcal{C}$, and the corresponding set of events $\mathcal{E}_u$, which we shall denote as just $\mathcal{E}$ from now on, for convenience.

The main source of difficulty in solving the gossip problem is the fact that the processes in $\mathcal{P}$ need to compute global information about the computation (u, ϕ) while each process only has access to a local, "partial" view of u. Although partial views of (u, ϕ) correspond to subsets of $\mathcal{E}$, not every subset of $\mathcal{E}$ arises from such a partial view. Those subsets of $\mathcal{E}$ which do correspond to partial views of (u, ϕ) are called ideals.

Ideals A set of events $I \subseteq \mathcal{E}$ is called an *order ideal* if I is closed with respect to $\sqsubseteq^*$—i.e., $e \in I$ and $f \sqsubseteq^* e$ implies $f \in I$ as well. We shall always refer to order ideals as just *ideals*.

The requirement that an ideal be closed with respect to $\sqsubseteq^*$ guarantees that the observation it represents is "consistent"—whenever an event e has been observed, so have all the events in the computation which necessarily precede e. Clearly the entire set $\mathcal{E}$ is an ideal, as is $e\!\downarrow$ for any $e \in \mathcal{E}$. It is easy to see that if I and J are ideals, so are $I \cup J$ and $I \cap J$.

EXAMPLE: *In Figure 1, the set* $I = \{(1, s_{p\to q}), (2, s_{p\to r}), (4, r_{r\leftarrow p}), (5, s_{r\to q}), (6, r_{q\leftarrow r})\}$ *is an ideal. However, the set* $J = \{(1, s_{p\to q}), (2, s_{p\to r}), (3, s_{p\to r}), (5, s_{r\to q}), (6, r_{q\leftarrow r})\}$ *is not an ideal, since* $(4, r_{r\leftarrow p}) \sqsubseteq^* (5, s_{r\to q})$ *but* $(4, r_{r\leftarrow p}) \notin J$.

We need to generalize the notion of $max_p(\mathcal{E})$, the maximum p-event in $\mathcal{E}$, to all ideals $I \subseteq \mathcal{E}$.

p-views For an ideal I, the $\sqsubseteq^*$-maximum p-event in I is denoted $max_p(I)$, provided $\#_{\mathcal{C}_p}(I) > 0$. The p-view of I is the set $I_p = max_p(I)\!\downarrow$. So, I_p is the set of all events in I which p can "see". (By convention, if $max_p(I)$ is undefined—i.e., if there is no p-event in I—the p-view I_p is the empty set.)

EXAMPLE: *In Figure 1, consider the ideal* $I = \{(1, s_{p\to q}), (2, s_{p\to r}), (4, r_{r\leftarrow p}), (5, s_{r\to q}), (6, r_{q\leftarrow r})\}$ *Then* I_p*, the p-view of* I *is* $\{(1, s_{p\to q}), (2, s_{p\to r})\}$ *whereas* I_q*, the q-view of* I*, is the entire ideal* I.

3 Primary information

For processes $p, q \in \mathcal{P}$, we have already defined $latest_{p\leftarrow q}(\mathcal{E})$, the latest information that p has about q after (u, ϕ). We can extend this definition to arbitrary ideals.

Let $I \subseteq \mathcal{E}$ be an ideal and $p, q \in \mathcal{P}$. Then $latest_{p\leftarrow q}(I)$ denotes the $\sqsubseteq^*$-maximum q-event in I_p, provided $\#_{\mathcal{C}_q}(I_p) > 0$. So, $latest_{p\leftarrow q}(I)$ is the latest q-event in I that p knows about. (As usual, if there is no q-event in I_p, the quantity $latest_{p\leftarrow q}(I)$ is undefined.)

It is clear that $latest_{p\leftarrow q}(I)$ will always correspond to a send action from $\mathcal{C}_q$. However $latest_{p\leftarrow q}(I)$ need not be of the form $s_{q\to p}$; the latest information that p has about q in I may have been obtained indirectly.

To maintain and update the latest information of processes, we need to expand the set of events that each process keeps track of. This expanded set will be called primary information. The primary information of a process contains not only its latest information about every other process but also information about acknowledged and unacknowledged messages in the system.

Message acknowledgments Let $I \subseteq \mathcal{E}$ be an ideal and $e \in I$ an event of the form $s_{p\to q}$. Then, e is said to have been *acknowledged* in I if $\phi^{-1}(e)$ exists and, moreover, belongs to I_p. Otherwise, e is said to be *unacknowledged* in I.

Notice that it is not enough for a message to have been received in I to deem it to be acknowledged. We demand that the event corresponding to the receipt of the message be "visible" to the sending process.

For an ideal I and a pair of processes p, q, let $received_{q \leftarrow p}(I)$ be the most recent message from p which q has received in I. In other words, $received_{q \leftarrow p}$ is the $\sqsubseteq^*$-maximum $s_{p \to q}$ event e in I such that $\phi^{-1}(e)$ belongs to I. Since (u, ϕ) is a fifo computation, all messages corresponding to $s_{p \to q}$ events below e must have also been received by q.

Let $unack_{p \to q}(I)$ be the set of unacknowledged $s_{p \to q}$ events in I. By definition, this set consists of all $s_{p \to q}$ events in I which lie above $received_{q \leftarrow p}(I_p)$. Since the underlying computation (u, ϕ) is B-bounded, $unack_{p \to q}(I)$ never contains more than B events.

It will be convenient to define the notion of primary information with respect to ideals rather than processes.

Primary information Let $I \subseteq \mathcal{E}$ be an ideal. The *primary information* of I, $primary(I)$, consists of the following events in I:

- The set $latest(I) = \{max_p(I) \mid p \in \mathcal{P}\}$.
- The collection of sets $unack(I) = \{unack_{p \to q}(I) \mid p, q \in \mathcal{P}\}$.
- The set $received(I) = \{received_{q \leftarrow p}(I) \mid p, q \in \mathcal{P}\}$.

Observe that we can treat $unack(I)$ to be just a set of events, though we have defined it as a collection of sets. For each event $e = (i, u(i))$ in $unack(I)$, the action $u(i)$ immediately determines the particular set $unack_{p \to q}(I)$ to which e belongs—$e \in unack_{p \to q}(I)$ iff $u(i) = s_{p \to q}$. Similarly, we can regard $latest(I)$ and $received(I)$ as just sets of events. For each event $e = (i, u(i))$ in $latest(I)$, $e = max_p(I)$ iff $u(i) \in \mathcal{C}_p$. In the same way, for each event $(i, u(i))$ in $received(I)$, $e = received_{q \leftarrow p}(I)$ iff $u(i) = s_{p \to q}$.

On the other hand, $primary(I)$ is an *indexed* set of events. If e belongs to $primary(I)$ we also need to record which of the three components e belongs to. However, often we shall abuse notation and treat $primary(I)$ as just a set of events. It is clear that $primary(I)$ contains only a bounded number of events.

Let $I \subseteq \mathcal{E}$ be an ideal and p a process such that $I_p \neq \emptyset$. Then, $primary(I_p)$ denotes the primary information of p in I—i.e., p's primary information is just the primary information of the p-view of I. Clearly, the "latest information" of p after I is contained in its primary information—for every process q, $latest_{p \leftarrow q}(I)$ is just $max_q(I_p)$.

To compare and update primary information, processes also need to remember how their primary events are ordered by $\sqsubseteq^*$.

Primary graph Let $I \subseteq \mathcal{E}$. The *primary graph* of I, $primary\text{-}graph(I)$ is the directed graph (V, E) where:

- $V = primary(I)$ (where $primary(I)$) is represented as an indexed set of events.)
- For $v_1, v_2 \in V$, let e_1 and e_2 be the corresponding events from I. Then, $(v_1, v_2) \in E$ iff $e_1 \sqsubseteq^* e_2$.

As with primary information, the primary graph of a process p in I is just the graph *primary-graph*(I_p).

For the moment we shall ignore the issue of assigning bounded time-stamps to events and assume that events are assigned unambiguous labels by some mechanism. For instance, as we mentioned earlier, each process could maintain a local counter and assign an increasing sequence of unique time-stamps to the events that it participates in.

Our first goal is to exhibit a procedure by which processes update their primary graphs *without* relying on the temporal order of events. Processes will only utilise the information about causality recorded in the primary graphs. All comparisons and updates of primary information will based purely on equality of event labels. This feature will allow us to extend the algorithm smoothly to the case where processes reuse labels.

4 Comparing primary information

Let $\mathcal{E}$ be the events corresponding to a computation (u, ϕ). Then, each ideal $I \subseteq \mathcal{E}$ corresponds to a possible partial computation of (u, ϕ). Let us assume that at the end of any partial computation I, each process maintains the information *primary-graph*(I_p).

In general, for an ideal $I \subseteq \mathcal{E}$, each pair of processes p and q will have incomparable information about I. The events known to both p and q lie in the ideal $I_p \cap I_q$. Events lying "above" the intersection are known to only one of the processes.

Suppose that q receives a message from p during the computation. Then, we have an event $e_p \in \mathcal{E}$ of the form $s_{p \rightarrow q}$ and a corresponding event $e_q \in \mathcal{E}$ of the form $r_{q \leftarrow p}$ such that $\phi(e_q) = e_p$.

Let $e'_q = max_q(e_q\downarrow - \{e_q\})$—i.e., e'_q is the maximum q-event strictly below e_q. So, $e'_q\downarrow$ represents the state of q's knowledge before receiving this message from p. Let I be the ideal $e_p\downarrow \cup e'_q\downarrow$.

For the moment, let us assume that $max_p(e'_q\downarrow) \sqsubset^+ e_p$—i.e., the message sent at e_p has "new" information for q about the state of p. Then, $e_p\downarrow = I_p$ and $e'_q\downarrow = I_q$. The message sent by p at e_p contains the primary graph *primary-graph*(I_p). On the other hand, before receiving this message, q's primary graph corresponds to *primary-graph*(I_q).

Our first observation is that if q knows both *primary-graph*(I_p) and *primary-graph*(I_q), it can determine which events in the two primary graphs lie within $I_p \cap I_q$ and which lie outside this intersection.

Lemma 1 *Let $I \subseteq \mathcal{E}$ be an ideal and p, q a pair of processes. Then, for each maximal event e in $I_p \cap I_q$, either $e \in latest(I_p) \cap unack(I_q)$ or $e \in unack(I_p) \cap latest(I_q)$.*

Proof Let e be a maximal event in $I_p \cap I_q$. Suppose it is an r-event, for some $r \in \mathcal{P}$. The event e must have $\sqsubset$-successors in both I_p and I_q. However, observe that any event f in $\mathcal{E}$ can have at most two immediate $\sqsubset$-successors—one "internal" successor within the process and, if f is a send event, one "external" successor corresponding to the matching receive event.

Thus, the maximal event e will have a $\lhd_{rr}$ successor e_r and a $\lhd_{rs}$ successor e_s, corresponding to some $s \in \mathcal{P}$. Assume that $e_r \in I_q - I_p$ and $e_s \in I_p - I_q$. Since the r-successor of e is outside I_p, $e = max_r(I_p)$. So e belongs to $latest(I_p)$. On the other hand, e is an unacknowledged $s_{r \to s}$ event in I_q. So, $e \in unack_{r \to s}(I_q)$, which is part of $unack(I_q)$.

Symmetrically, if $e_r \in I_p - I_q$ and $e_s \in I_q - I_p$, we find that e belongs to $unack(I_p) \cap latest(I_q)$. □

Thus, when q receives p's primary graph, q can collect together in a set M all the events that lie in $latest(I_p) \cap unack(I_q)$ and $unack(I_p) \cap latest(I_q)$. By the preceding lemma, the events in M subsume the maximal events in the intersection $I_p \cap I_q$. (It is easy to see that those events in M which are not actually maximal still lie within the intersection.)

The process q can use M to check whether a primary event $e \in primary(I_p) \cup primary(I_q)$ lies inside or outside the intersection—e lies inside the intersection iff it lies below one of the elements in M. These comparisons can be made using the edge information in the graphs $primary\text{-}graph(I_p)$ and $primary\text{-}graph(I_q)$.

Now, it is easy for q to compare the events in $latest(I_p)$ with those in $latest(I_q)$ to determine which of p and q have more recent information about every other process r.

Lemma 2 *Let $I \subseteq \mathcal{E}$ be an ideal and p, q a pair of processes. Let $e = max_r(I_p)$ and $f = max_r(I_q)$ such that $e \neq f$. Then, $e \sqsubset^+ f$ iff $f \in I_q - I_p$.*

Proof ($\Rightarrow$) Suppose that $e \sqsubset^+ f$. If f belongs to I_p then $e \neq max_r(I_p)$, which is a contradiction. On the other hand, f clearly belongs to I_q, so $f \in I_q - I_p$.

($\Leftarrow$) Suppose that $f \in I_q - I_p$. If $f \sqsubset^+ e$, then $f \in I_p$ since I_p is an ideal. This is a contradiction. So it is not the case that $f \sqsubset^+ e$. Since $f \neq e$ and all r-events are totally ordered by $\sqsubseteq^*$, we must have $e \sqsubset^+ f$. □

Once q has compared all events of the form $max_r(I_p)$ and $max_r(I_q)$, it can easily update its sets $unack_{r \to s}(I_q)$, where $r \neq q$. The process which has better information about r also has better information about unacknowledged events of the form $s_{r \to s}$ in I. In other words, q inherits the sets $unack_{r \to s}(I_p)$ for every process r such that $max_r(I_p)$ is more recent than $max_r(I_q)$. On the other hand, if $max_r(I_p)$ is older than $max_r(I_q)$, then q ignores p's sets $unack_{r \to s}(I_p)$ since it already has better information about these events.

At this stage, using the data in $primary\text{-}graph(I_p)$ and $primary\text{-}graph(I_q)$, q has updated all of $primary(e_q\downarrow)$ except for the sets $\{unack_{q \to r}(e_q\downarrow)\}_{r \in \mathcal{P}}$ and $received(e_q\downarrow)$. (Recall that e_q was the event where q received p's message sent at e_p.)

We first describe how to construct $received(e_q\downarrow)$. Clearly $received_{q \leftarrow p}(e_q\downarrow) = e_p$. To fill in the rest of $received(e_q\downarrow)$, we need the following observation, which we state without proof.

Proposition 3 *For every pair of processes (r, s) such that $(r, s) \neq (p, q)$,*

$$received_{s \leftarrow r}(e_q\downarrow) = \begin{cases} received_{s \leftarrow r}(I_p) & \text{if } max_s(I_q) \sqsubset^+ max_s(I_p) \\ received_{s \leftarrow r}(I_q) & \text{otherwise} \end{cases}$$

So, once q has compared the events in $latest(I_p)$ and $latest(I_q)$, it can decide which of the events in $received(e_q\downarrow)$ are inherited from $received(I_p)$ and which are retained from $received(I_q)$.

Process q can now use the information in $received(e_q\downarrow)$ to update the sets $\{unack_{q\to r}(I_q)\}_{r\in\mathcal{P}}$ by purging acknowledged events from these lists. Formally, for every process r, $unack_{q\to r}(e_q\downarrow)$ consists of all the events in $unack_{q\to r}(I_q)$ which lie above $received_{r\leftarrow q}(e_q\downarrow)$. This update can be made with the edge information in $primary\text{-}graph(I_q)$.

Having constructed the sets $latest(e_q\downarrow)$, $unack(e_q\downarrow)$ and $received(e_q\downarrow)$, we need to extend this set to the graph $primary\text{-}graph(e_q\downarrow)$.

Let $f_1, f_2 \in primary(e_q\downarrow)$. If both f_1 and f_2 came from $primary(I_p)$, then we add an edge from f_1 to f_2 in $primary\text{-}graph(e_q\downarrow)$ iff a corresponding edge existed in $primary\text{-}graph(I_p)$. A symmetric situation applies if both f_1 and f_2 were contributed by $primary(I_q)$.

So, the only interesting case is when f_1 and f_2 originally came from different processes. Without loss of generality, suppose that f_1 came from $primary(I_p)$ and f_2 from $primary(I_q)$. Then, from the method which we used to compare events, we know that f_1 must have been in $I_p - I_q$ and f_2 must have been in $I_q - I_p$. So, it is clear that f_1 and f_2 are unordered in $\mathcal{E}$ and there is therefore no edge between them in $primary\text{-}graph(e_q\downarrow)$.

So far, we have shown how to construct $primary\text{-}graph(e_q\downarrow)$ assuming that $max_p(e'_q\downarrow) \sqsubset^+ e_p$. If q already has better information about p—i.e., $e_p \sqsubseteq^* max_p(e'_q\downarrow)$—it is easy to see that e_p must in fact lie strictly below $max_p(e'_q\downarrow)$, since q's knowledge of e_p must have come from a message sent by p *after* e_p to some other process r. In this case, q can no longer use Lemma 1 to compute the maximal elements in the intersection $e_p\downarrow \cap e'_q\downarrow$ since the p-view of the ideal $I = e_p\downarrow \cup e'_q\downarrow$ is *not* $e_p\downarrow$ but $max_p(e'_q\downarrow)\downarrow$.

However, it is easy to see that if $e_p \sqsubset^+ max_p(e'_q\downarrow)$ then $e_p\downarrow \subseteq e'_q\downarrow$. So, for every other process r, $max_r(e_p\downarrow) \sqsubseteq^* max_r(e'_q\downarrow)$. It then follows that the only update that q has to make to $primary\text{-}graph(e'_q\downarrow)$ is to set $received_{q\leftarrow p}(e_q\downarrow)$ to e_p. The rest of $primary\text{-}graph(e_q\downarrow)$ is inherited from $primary\text{-}graph(e'_q\downarrow)$.

All we need is a means for q to detect whether e_p is a "new" message. Based on this, q can decide whether to ignore the information in $primary\text{-}graph(e_p\downarrow)$. This is not very difficult because of the following observation.

Proposition 4 *Let e_p be a $s_{p\to q}$ event such that $\phi^{-1}(e_p) = e_q$ and let $e'_q = max_q(e_q\downarrow - \{e_q\})$. Then $e_p \sqsubseteq^* max_p(e'_q\downarrow)$ iff $e_p \in unack_{p\to q}(e'_q\downarrow)$.*

In other words, to check if the message is old, all q has to do is to see if e_p is already in its set $unack(e'_q\downarrow)$. This leads to the following general statement which summarizes the results of this section.

Lemma 5 *Let e_p be a $s_{p\to q}$ event in $\mathcal{E}$ such that $\phi^{-1}(e_p) = e_q$. Let $e'_q = max_q(e_q\downarrow - \{e_q\})$. Then, q can construct $primary\text{-}graph(e_q\downarrow)$ from the graphs $primary\text{-}graph(e_p\downarrow)$ and $primary\text{-}graph(e'_q\downarrow)$.*

5 Bounded time-stamps

To make the protocol described in the previous section effective, we have to bound the amount of information recorded in the primary graph of each process

by limiting the size of the labels used to identify events.

Notice that the procedure for updating primary graphs only checks the labels of events which actually lie in the primary graphs of the sending and receiving processes. Call an event e "current" in I if e belongs to $primary(I_p)$ for some process p.

Let N be the number of processes in the system. Since the underlying computation is B-bounded, we know that there are at most $N + (B+1)N^2$ distinct events in $primary(I_p)$ for process p—there are at most N events in $latest(I_p)$, at most B events in each of the sets $unack_{q \to r}(I_p)$ and at most N^2 events overall in $received(I_p)$. So, at any given time, the number of events across the system which are current is bounded by $N(N + (B+1)N^2)$.

From the way primary information is defined, it is clear that all current events are actually send events. Each send event $s_{p \to q}$ begins by being current—the moment the message is sent, the event is added to the list of unacknowledged messages from p to q. Eventually, q acknowledges this message and p gets rid of it from its unacknowledged list. Meanwhile, as the computation progresses, this event may get added to the primary information of other processes. However, gradually it recedes into the past, until it drops out of the primary information of *all* processes. At this time, the label assigned to this event can be reused—the old event with the same label can *never* become current again.

Processes can keep track of which events in the system are current by maintaining one additional level of data, called secondary information.

Secondary information Let I be an ideal. The *secondary information* of I is the collection of (indexed) sets $primary(e\downarrow)$ for each event e in $primary(I)$. This collection of sets is denoted $secondary(I)$.

The following lemma says that the only p-events which can be current in the system are those which occur in p's secondary information.

Lemma 6 *Let $I \subseteq \mathcal{E}$ be an ideal and e a p-event which belongs to $primary(I_q)$ for some process q. Then, $e \in secondary(I_p)$.*

A proof of this lemma can be found in the full paper [8]. We will use the preceding result in the following form.

Corollary 7 *Let e be a p-event such that $e \notin secondary(I_p)$. Then $e \notin primary(I_q)$ for any $q \in \mathcal{P}$.*

Our update procedure does not rely on the temporal order of events. So long as all processes which refer to the same label in their primary information are actually talking about the same event, reusing labels should cause no confusion. Therefore, if p knows that no p-event labelled ℓ is currently part of the primary information of any process in the system, it can safely use ℓ to time-stamp the next message which it sends.

Secondary information can be updated in a straightforward manner when we update primary information—if q inherits an event e from p's primary information, it also inherits the secondary information $primary(e\downarrow)$ associated with e. Notice that it suffices to maintain secondary information as an indexed set—we do not need to maintain secondary *graphs* as we do primary graphs.

This at once gives us a protocol which solves the gossip problem for B-bounded fifo computations.

The gossip protocol

Let $\mathcal{L}$ be a finite set of labels of such that $|\mathcal{L}| > (N^2+(B+1)N^3)$. All processes use the set $\mathcal{L}$ to time-stamp messages. Each process p maintains its primary graph $\mathit{primary\text{-}graph}_p = (V_p, E_p)$, where V_p consists of the (indexed) sets of labels latest_p, unack_p and $\mathit{received}_p$.

A typical element of latest_p is a pair of the form (ℓ, q)—this will mean that the maximum q-event known to p is time-stamped ℓ. Similarly, elements of unack_p and $\mathit{received}_p$ are triples. An entry (ℓ, q, r) in unack_p signifies that, as far as p knows, the $s_{q\to r}$ event labelled ℓ has not been acknowledged. In the same vein, a typical entry (ℓ, q, r) in $\mathit{received}_p$ denotes that, as far as p knows, the most recent message from q to r which has actually been delivered is time-stamped ℓ.

Finally, the process p maintains its secondary information $\mathit{secondary}_p$ as an indexed set of labels. If $\bar{e}$ is a tuple from $\mathit{latest}_p \cup \mathit{unack}_p \cup \mathit{received}_p$, then an event in $\mathit{latest}(\bar{e}\downarrow)$ will be represented as $(\ell', r, \bar{e})$, indicating that the maximum r-event in $\bar{e}\downarrow$ is time-stamped ℓ'. In a similar manner, an entry $(\ell', r, s, \bar{e})$ in $\mathit{unack}(\bar{e}\downarrow)$ signifies that there is a $s_{r\to s}$ event time-stamped ℓ' which is unacknowledged within $\bar{e}\downarrow$. And, finally, an entry $(\ell', r, s, \bar{e})$ in $\mathit{received}(\bar{e}\downarrow)$ means that the most recent message from r to s which has been delivered within $e\downarrow$ is timestamped ℓ'.

Initially, for each p, latest_p, unack_p, $\mathit{received}_p$ and $\mathit{secondary}_p$ are empty.

Sending a message When p sends a message to q it does the following:

- Choose a label ℓ from $\mathcal{L}$ which does not appear as the first component of any tuple in $\mathit{latest}_p \cup \mathit{unack}_p \cup \mathit{received}_p \cup \mathit{secondary}_p$.
- Remove the old event (ℓ', p) from latest_p, if it exists. Also remove all associated events from $\mathit{secondary}_p$—i.e., tuples of the form (ℓ'', p', ℓ', p) and $(\ell'', p', p'', \ell', p)$.
- Add (ℓ, p, q) to unack_p and (ℓ, p) to latest_p. Add an edge in E_p from each tuple in $\mathit{latest}_p \cup \mathit{unack}_p \cup \mathit{received}_p$ to the new tuples $(\ell, p, q) \in \mathit{unack}_p$ and $(\ell, p) \in \mathit{latest}_p$.
- For each pair (ℓ', p') in latest_p, add (ℓ', p', ℓ, p) to $\mathit{secondary}_p$. Similarly, for each triple (ℓ', p', p'') in $\mathit{unack}_p \cup \mathit{received}_p$, add $(\ell', p', p'', \ell, p)$ to $\mathit{secondary}_p$.
- Send $\mathit{primary\text{-}graph}_p$ and $\mathit{secondary}_p$ to q.

Receiving a message On receiving a message from p, q does the following:

- Extract the label ℓ of the new message.
- Replace the current triple (ℓ', p, q) in $\mathit{received}_q$, if it exists, by (ℓ, p, q).
- If (ℓ, p, q) does not already belong to unack_q then update $\mathit{primary\text{-}graph}_q$ and $\mathit{secondary}_q$ by comparing $\mathit{primary\text{-}graph}_p$ and $\mathit{secondary}_p$ in the message with $\mathit{primary\text{-}graph}_q$ and $\mathit{secondary}_q$ currently maintained by q.

So, on sending a message, p chooses a label ℓ which is not currently in use in the system and uses ℓ to time-stamp the message. It then replaces the latest p label in $latest_p$ by ℓ and also adds ℓ to the list of unacknowledged $s_{p \to q}$ events. Finally, it places the new event at the "top" of its primary graph and sets the secondary information with respect to the new event to be its overall primary information. It then sends its new data structures $primary\text{-}graph_p$ and $secondary_p$ to q.

When q receives the message labelled ℓ, it first records in $received_q$ that the most recently received message from p is labelled ℓ. (Extracting the label of the message can be done by looking, for instance, for the E_p^*-maximal event in $primary\text{-}graph_p$). It then checks whether the message is new. If so, it updates its primary graph and secondary information following the results in Lemmas 1 and 2 and Proposition 3.

Message complexity

Lemma 8 *Let N be the number of processes in the system. For each process p, the information in $primary\text{-}graph_p$ and $secondary_p$ requires at most $O(N^4 log N)$ bits to write down.*

Proof We know that $|\mathcal{L}|$ is $O(N^3)$. So each label in $\mathcal{L}$ can be written down using $O(\log N)$ bits. Similarly, each process name can be written down using $O(\log N)$ bits. So each tuple in the sets $latest_p$, $unack_p$, $received_p$ and $secondary_p$ requires only $O(\log N)$ bits to write down.

There are $O(N^2)$ entries in $primary_p$. So, the edge relation E_p of $primary\text{-}graph_p$ can be represented in terms of an adjacency matrix, using $O(N^4)$ bits. So, all of $primary\text{-}graph_p$ can be written down in $O(N^4)$ bits.

The real bottleneck turns out to be $secondary_p$. There are $O(N^2)$ elements in $primary_p$. For each of these elements $\bar{e}$, we have to maintain $primary(\bar{e}\downarrow)$, which requires $O(N^2 \log N)$ bits. So, overall $secondary_p$ requires $O(N^4 \log N)$ bits to write down.

□

Putting together all the results we have proved so far, we can state the following theorem

Theorem 9 *The protocol we have described solves the gossip problem for fifo, B-bounded computations with only a bounded amount of additional information being added to each message of the underlying computation.*

6 Non-fifo computations

We briefly describe how to extend our protocol to deal with B-bounded computations which are *not* fifo. We do not go into all the formal details here due to a lack of space. Details are available in the full paper [8].

The essential difference is in the procedure for collecting acknowledgements. In a non-fifo computation, messages may be received out of order. So, we cannot infer from the fact that $e = received_{q \leftarrow p}(I)$ that all messages sent by p to q

before e have actually reached q. In general, we have to explicitly acknowledge each message label to the sending process. This can be done by recording for each pair of processes p, q, the set $\mathit{received\text{-}set}_{q \leftarrow p}(I)$ consisting of all events e of the form $s_{p \rightarrow q}$ such that $\phi^{-1}(e)$ belongs to I.

So, we extend the information maintained by each process p to include the sets $\mathit{received\text{-}set}_{r \leftarrow q}(I_p)$ for each pair of processes q, r. These sets can be updated using an extension of Proposition 3. The problem is, of course, that these sets grow without bound as the computation proceeds.

However, it can be shown that once e is added to $\mathit{received\text{-}set}_{q \leftarrow p}(I_q)$ by q, it will become known to p (i.e., it will appear in $\mathit{received\text{-}set}_{q \leftarrow p}(I'_p)$ for $I' \supseteq I$) within a bounded number of communications in the system. This follows from the assumption that the underlying computation is B-bounded. Once e appears in $\mathit{received\text{-}set}_{q \leftarrow p}(I'_p)$, it has been acknowledged and can be removed from q's list of received messages.

As a result, it turns out that each process p needs to retain only the B-most recent events in $\mathit{received\text{-}set}_{r \leftarrow q}(I_p)$, where "most recent" refers to the order in which the messages were received by r and not the order in which they were sent by q. So, acknowledgments can still be propogated using a bounded amount of information, even when messages arrive out of order and our protocol can be extended to solve the gossip problem for arbitrary B-bounded computations.

References

[1] R. Cori, Y. Metivier: Approximations of a trace, asynchronous automata and the ordering of events in a distributed system, *Proc. ICALP '88*, *LNCS* **317** (1988) 147–161.

[2] R. Cori, Y. Metivier, W. Zielonka: Asynchronous mappings and asynchronous cellular automata, *Inform. and Comput.*, **106** (1993) 159–202.

[3] D. Dolev, N. Shavit: Bounded concurrent time-stamps are constructible, *Proc. ACM STOC* (1989) 454–466.

[4] A. Israeli, M. Li: Bounded time-stamps, *Proc. 28th IEEE FOCS* (1987) 371–382.

[5] R. Krishnan, S. Venkatesh: Optimizing the gossip automaton, *Report TCS-94-3*, School of Mathematics, SPIC Science Foundation, Madras, India (1994).

[6] L. Lamport: Time, clocks and the ordering of events in a distributed system, *Comm. ACM* **17**(8) (1978) 558–565.

[7] L. Lamport, N. Lynch: Distributed Computing: Models and Methods, in: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science: Volume B*, North-Holland, Amsterdam (1990) 1157–1200.

[8] M. Mukund, K. Narayan Kumar, M. Sohoni: Keeping track of the latest gossip in message-passing systems, *Report TCS-95-3*, School of Mathematics, SPIC Science Foundation, Madras, India (1995).

[9] M. Mukund, M. Sohoni: Keeping track of the latest gossip: Bounded time-stamps suffice, *Proc. FST&TCS '93*, *LNCS* **761** (1993) 388–399.