Algorithms for Extracting Timeliness Graphs[⋆]

Carole Delporte-Gallet[1], Stéphane Devismes[2], Hugues Fauconnier[1], and Mikel Larrea[3]

[1] Université Paris Diderot
LIAFA
{Carole.Delporte,Hugues.Fauconnier}@liafa.jussieu.fr
[2] Université Joseph Fourier, Grenoble I
VERIMAG UMR 5104
Stephane.Devismes@imag.fr
[3] University of the Basque Country, UPV/EHU
Mikel.Larrea@ehu.es

Abstract. We consider asynchronous message-passing systems in which some links are timely and processes may crash. Each run defines a *timeliness graph* among correct processes: (p, q) is an edge of the timeliness graph if the link from p to q is timely (that is, there is a bound on communication delays from p to q). The main goal of this paper is to approximate this timeliness graph by graphs having some properties (such as being trees, rings, ...). Given a family S of graphs, for runs such that the timeliness graph contains at least one graph in S then using an *extraction algorithm*, each correct process has to converge to the same graph in S that is, in a precise sense, an approximation of the timeliness graph of the run. For example, if the timeliness graph contains a ring, then using an extraction algorithm, all correct processes eventually converge to the same ring and in this ring all nodes will be correct processes and all links will be timely.

We first present a general extraction algorithm and then a more specific extraction algorithm that is communication efficient (*i.e.*, eventually all the messages of the extraction algorithm use only links of the extracted graph).

1 Introduction

We consider partially synchronous models like in [6] or [7] in which some processes may crash. In such systems some links are timely, meaning that the communication delays are bounded [2], and some other are not. Generally, these timeliness properties of links have been used to solve the consensus problem as in [7] or to implement failure detectors like Ω that realizes an eventual election of a correct process (*e.g.*, [1, 4, 10–12]). In this paper we are more specifically interested in detecting the timeliness of the links in order to approximate the

[⋆] This work has been supported in part by the ANR project *SHAMAN*, the INRIA project GANG, and the Basque Government (grant MV-2009-1-10).

timeliness relation on links in each run. If processes are able to eventually determine which links are timely, then avoiding to use non timely links could help to improve the efficiency of the communication that can be particularly interesting for routing algorithms.

More precisely, each run of the system eventually converges to a *timeliness graph* whose nodes are the correct processes and directed edges are the timely edges among correct processes, and an *extraction algorithm* is an algorithm such that all correct processes eventually agree on an identical graph that approximates the timeliness graph.

For example, assume the system ensures that there is at least one correct process that communicates in a timely way with all other processes, such a process is an *eventual source* [2] and it could be interesting for the processes to choose and agree on such an eventual source. This way, we not only realize an eventual leader election but also the chosen leader is able to communicate in a timely way with the rest of correct processes.

If we assume now that instead of an eventual source there is an eventual *root* in the system, that is a correct process that may communicate with every process by a communication path using only timely links, then choosing and agreeing on such an eventual root realizes an eventual leader election (the root is the eventual leader) but this also enables to ensure a routing of all messages from the root to any other processes using only timely links.

In the same way, if the system ensures that there is always a cycle containing all correct processes in the timeliness graph of the run, then choosing and agreeing on one such cycle enables to eventually build a ring between all correct processes that uses only timely links. Note that in this case the processes eventually agree on the list of all correct processes too, as a consequence we obtain a failure detector $\Diamond P$ [5].

More precisely, consider some structural property $\mathcal{P}$ of graphs (like being a star, a ring, a tree, a complete graph...). An algorithm *extracting* a graph G verifying $\mathcal{P}$ has to ensure that (1) all the correct processes eventually agree on G, (2) all the correct processes are nodes of G and (3) G is an "approximation" of the timeliness relation of the run. Actually, "approximation" means that the subgraph of G induced by the correct processes is obtained from a directed cut (dicut) [4] of G and is a subgraph of the timeliness graph.

Contributions. In this paper, we first introduce and specify the problem of extraction of graphs in some set $\mathcal{X}$. We consider only systems in which a solution may exist: in all runs there is at least one graph in $\mathcal{X}$ that is compatible with the run.

We prove that this problem cannot be solved for some set of graphs and we give a sufficient condition on the set of graphs to be extracted. This condition is rather simple: the set of graphs has to be closed by directed cut reduction. Then, we give an extraction algorithm for every set of graphs verifying this property.

[4] A directed cut (X, Y) of directed graph $G = \langle N, E \rangle$ is a partition of N such that there is no directed edge from Y to X.

Moreover, if the graphs in $\mathcal{X}$ are all strongly connected, the algorithm gives an exact extraction, that is, the set of nodes of the extracted graph is exactly the set of correct processes of the run. Reciprocally, we show that there exist sets of graphs that admit extraction but no exact extraction.

Besides, we show that finding an approximation is even so interesting: in the extracted graph any path between a pair of correct processes is only constituted of timely links. Hence, the approximation can be used to timely route messages, *e.g.*, in the previous example with a root, the approximation will give us a tree whose root is a correct process and with a path containing only correct processes from the root to every correct process.

One drawback of this algorithm is the fact that forever all correct processes have to send messages on all links. Hence if k is the number of correct processes $k(k-1)$ links will be used forever by the extraction algorithm. We are then interested in *communication efficient* [11] implementations of the extraction problem. That is, eventually all correct processes only send messages along the edges of the extracted graph. For example, consider the example of a system with a timeliness ring, eventually only $k-1$ links of the system are used. We propose an efficient extraction algorithm for sets of graphs containing at least one correct process with directed paths from this process to all correct processes.

Roadmap. In the next section, we define the model used in this paper and present some examples of systems. In Section 3, we define the extraction problem and give some of its properties. Our two algorithms are presented in Sections 4 and 5, respectively. Finally, we make some concluding remarks in Section 6.

Due to the lack of space, some technical proofs have been omitted. For further details, see the online technical report [8].

2 Informal Model

Graphs. We begin with some definitions and notations concerning graphs. For a directed graph $G = \langle N, E \rangle$, $Node(G)$ and $Edge(G)$ denote N and E, respectively. Given a graph G and a set $M \subseteq Node(G)$, $G[M]$ is the *subgraph* of G induced by M, *i.e.*, $G[M]$ is the graph $\langle M, Edge(G)[M] \rangle$ where $(p,q) \in Edge(G)[M]$ if and only if $p, q \in M$ and $(p,q) \in Edge(G)$.

The tuple (X, Y) is a *directed cut* (*dicut* for short) of G if and only if X and Y define a partition of $Node(G)$ and there is no directed edge $(y, x) \in Edge(G)$ such that $x \in X$ and $y \in Y$. We say that G' is a *dicut reduction* from G if there exists a dicut (X, Y) of G such that $G' = G[X]$. A set S of graphs is *dicut-closed* if and only if it is closed under dicut reduction, namely if $G \in S$ then all the graphs obtained by a dicut-reduction of G are in S.

Processes and Links. We consider distributed systems composed of n processes which communicate by message-passing through directed links. We denote the set of processes by $\Pi = \{p_1, ..., p_n\}$. We assume that the communication graph is complete, *i.e.*, for each pair of distinct processes (p, q), there is a directed link from p to q.

A process may fail by crashing, in which case it definitely stops its local algorithm. A process that never crashes is said to be *correct*, *faulty* otherwise.

The (directed) links are *reliable*, *i.e.*, every message sent through a link (p, q) is eventually received by q if q is correct and if a message m from p is received by q, m is received by q at most once, and only if p previously sent m to q.

The links being reliable, an implementation of the *reliable broadcast* [9] is possible. A reliable broadcast is defined with two primitives: $\texttt{rbroadcast}\langle m\rangle$ and $\texttt{rdeliver}\langle m\rangle$. Informally, after a correct process p invokes $\texttt{rbroadcast}\langle m\rangle$, all correct processes eventually $\texttt{rdeliver}\langle m\rangle$; after a faulty process p invokes $\texttt{rbroadcast}\langle m\rangle$, either all correct processes eventually $\texttt{rdeliver}\langle m\rangle$ or correct processes never $\texttt{rdeliver}\langle m\rangle$.

Timeliness. To simplify the presentation, we assume the existence of a discrete global clock. This is merely a fictional device: the processes do not have access to it. We take the range $\mathcal{T}$ of the clock's ticks to be the set of natural numbers.

We assume that every correct process p is *timely*, *i.e.*, there is a lower and an upper bound on the execution rate of p. Correct processes also have clocks that are not necessarily synchronized but we assume that they can accurately measure intervals of time.

A link (p, q) is *timely* if there is an unknown bound δ such that no message sent by p to q at time t may be received by q after time $t + \delta$.

A *timeliness graph* is simply a directed graph whose set of nodes are a subset of $\mathit{\Pi}$. The timeliness graph represents the timeliness properties of the links. Intuitively, for timeliness graph G, $Node(G)$ is the set of correct processes and (p, q) is in $Edge(G)$ if and only if the link (p, q) is timely.

Runs. An algorithm $\mathcal{A}$ consists of n deterministic (infinite) automata, one for each process; the automaton for process p is denoted $\mathcal{A}(p)$. The execution of an algorithm $\mathcal{A}$ proceeds as a sequence of process *steps*. Each process performs its steps atomically. During a step, a process may send and/or receive some messages and changes its state.

A run r of algorithm $\mathcal{A}$ is a tuple $r = \langle T, I, E, S\rangle$ where T is a timeliness graph, I is the initial state of the processes in $\mathit{\Pi}$, E is an infinite sequence of steps of $\mathcal{A}$, and S is a list of increasing time values indicating when each step in E occurred. A run must satisfy usual properties concerning sending and receiving messages. Moreover, we assume that (1) all correct processes make an infinite number of steps: $p \in Node(T)$ if and only if p makes an infinite number of steps in E and (2) the timeliness of links is deduced from the timeliness graph T: $(p, q) \in Edge(T)$ if and only if the link (p, q) is timely with respect to E and S.

In the following for run $r = \langle T, I, E, S\rangle$, $T(r)$ denotes T the timeliness graph of r, and $Correct(r)$ is the set of correct processes for the run r, namely, $Correct(r) = Node(T(r))$. Note that by definition, (p, q) is a timely link if and only if $(p, q) \in Edge(T)$.

Remark that in the definition given here a link may be timely even if no message is sent on the link. If link (p, q) is FIFO (*i.e.*, messages from p to q are received in the order they are sent) and p regularly sends messages to q, then the timeliness of these messages implies the timeliness of the link itself. So in the following we always assume that links are FIFO.

2.1 Some Systems

We say that timeliness graph G is *compatible with timeliness graph* G' if and only if (1) $Node(G) = Node(G')$ and (2) $Edge(G) \subseteq Edge(G')$. By extension, timeliness graph G is *compatible with run* r if G is compatible with $T(r)$, the timeliness graph of r. Hence, timeliness graph G is compatible with run r if $Node(G)$ is the set of correct processes in r and if (p,q) is an edge of G then (p,q) is timely in r.

A *system* $\mathcal{X}$ is defined as a set of timeliness graphs. The set of runs of system $\mathcal{X}$ denoted $R(\mathcal{X})$ is the set of all runs r such that there exists a timeliness graph G in $\mathcal{X}$ compatible with r.

Below, we define the systems considered in this paper:

- $\mathcal{ASYNC}$ is the set of all timeliness graphs G such that $Edge(G) = \emptyset$. In $\mathcal{ASYNC}$ there is no timeliness assumption about links and $R(\mathcal{ASYNC})$ is the set of all runs in an asynchronous system.
- $\mathcal{COMPLETE}$ is the set of all complete graphs whose nodes are the subsets of Π.
- $\mathcal{STAR}$ is the set of all timeliness graphs with a *source*, *i.e.*, $G \in \mathcal{STAR}$ if and only if $Node(G) \subseteq \Pi$ and there exists $p_0 \in Node(G)$ (the center of the star or the source) such that $Edge(G) = \{(p_0,q) | q \in Node(G) \setminus \{p_0\}\}$. Clearly a run r is in $R(\mathcal{STAR})$ if and only if there is at least one *source* in r.
- $\mathcal{TREE}$ is the set of all timeliness graphs G that are rooted directed trees, *i.e.*, $|Edge(G)| = |Node(G)| - 1$ and there exists p_0 in $Node(G)$ such that $\forall q \in Node(G)$, there is a directed path of G from p_0 to q. Clearly a run r is in $R(\mathcal{TREE})$ if and only if there is at least one timely path from a correct process to all correct processes.
- $\mathcal{RING}$ is the set of all timeliness graphs G such that G is a directed cycle (a ring). Clearly a run r is in $R(\mathcal{RING})$ if and only if there is a timely (directed) cycle over all correct processes.
- $\mathcal{SC}$ is the set of all timeliness graphs that are strongly connected. Clearly, a run r is in $R(\mathcal{SC})$ if and only if there exists a (directed) timely path between each pair of distinct correct processes.
- $\mathcal{BIC}$ is the set of all timeliness graphs G such that for all p, $q \in Node(G)$, there exist at least two distinct paths from p to q. $\mathcal{BIC}$ corresponds to the set of 2-strongly-connected graphs. Clearly, a run r is in $R(\mathcal{BIC})$ if and only if there exists at least two distinct timely paths between each pair of distinct correct processes.
- $\mathcal{PAIR}$ is the set of all timeliness graphs G such that $Edge(G) = \{(p,q), (q,p)\}$ with $p,q \in Node(G)$ and $p \neq q$. Clearly, a run r is in $R(\mathcal{PAIR})$ if and only if there exists two distinct correct processes p and q such that (p,q) and (q,p) are timely links.

3 Extraction Algorithms

Given a system $\mathcal{X}$, the goal of an *extraction algorithm* is to ensure that in each run r in $R(\mathcal{X})$, all correct processes eventually agree on the same element of $\mathcal{X}$ and that this element is, in some precise sense, an approximation of the timeliness graph of run r.

For example, in $\mathcal{RING}$, all processes have to eventually agree on some ring and this ring has to be compatible with the timeliness graph of the run. In particular this ring contains all the correct processes. However, the compatibility relation may be too strong: In many systems, it is not possible to distinguish between a crashed process and a correct one, so the graph G on which the processes eventually agree may contain crashed processes and then the graph is not exactly compatible with the run. Then we weaken the compatibility and impose only that the subgraph of G induced by the set of correct processes of the run is a dicut reduction of the timeliness graph of the run.

We now formally define what an extraction algorithm is. First, in such an algorithm, every process p maintains a local variable G_p which contains a timeliness graph. Then, we say that an algorithm *extracts a timeliness graph in* $\mathcal{X}$ if and only if for every run r in $R(\mathcal{X})$ there is a timeliness graph G (called the *extracted graph*) such that:

- *Convergence:* for all correct processes p there is a time t after which $G_p = G$
- *Compatibility:* $G[Correct(r)]$ is compatible with $T(r)$
- *Closure:* $G[Correct(r)]$ is a dicut reduction of G or is equal to G
- *Validity:* G is in $\mathcal{X}$

Remark that for all systems that contain $\mathcal{ASYNC}$ there is a trivial extraction algorithm: for each run processes extract the graph G such that $Node(G) = \Pi$ and $Edge(G) = \emptyset$.

A more constrained version of the extraction problem is the following: an algorithm $\mathcal{A}$ *extracts exactly* timeliness graphs in $\mathcal{X}$ if for every run r in $R(\mathcal{X})$, the extracted graph G is compatible with $T(r)$. In this case, all correct processes eventually know the exact set of correct processes: it is the set of nodes of the extracted graph.

Some Results about Extraction Algorithms. First we show that an extraction algorithm may help to route messages using only timely links:

Lemma 1. *Let G be a graph extracted from run r, if (p, q) is in $Edge(G)$ and q is a correct process then p is correct.*

Proof. By contradiction, assume that p is not correct, then $(Correct(r), Node(G) - Correct(r))$ is not a dicut because $(p, q) \in Edge(G)$, $p \in Node(G) - Correct(r)$ and $q \in Correct(r)$, which contradicts the Closure property.

From this lemma and the Compatibility property, we deduce directly:

Proposition 1. *If $(p = q_0, \ldots, q_i, \ldots, q = q_m)$ is a path in the extracted graph and p and q are correct processes, then for every i such that $0 \leq i < m$ the link (q_i, q_{i+1}) is timely and process q_i is correct.*

From a practical point of view, this proposition shows that the extracted graph may be used to route messages between processes using only timely links: the route from p to q is a path in the extracted graph (if any). All intermediate nodes are correct processes and agree on the extracted graph and then on the path.

For example with $\mathcal{TREE}$, the tree extracted by the algorithm enables to route messages from the root of the tree to any other processes and the routing uses only timely links.

Generally, the main goal of the extraction algorithm is not only to extract a graph G in $\mathcal{X}$ but also to ensure that $G[Correct(r)]$ is in $\mathcal{X}$ (even if the processes do not know the set of correct processes). In particular, this property is ensured if $\mathcal{X}$ is dicut-closed: the Closure property implies that $G[Correct(r)]$ is in $\mathcal{X}$.

Among the systems we consider, only system $\mathcal{PAIR}$ is not dicut-closed: $H = \langle \{x\}, \emptyset \rangle$ is a dicut reduction of $G = \langle \{x, y, z\}, \{(y, z), (z, y)\} \rangle$ but is not in $\mathcal{PAIR}$. It is easy to verify that every other previously introduced system is dicut-closed. For these systems we obtain:

Proposition 2. *Consider any extraction algorithm for the system $\mathcal{X}$.*

- *If $\mathcal{X} = \mathcal{STAR}$, then the center of the extracted star is a correct process.*
- *If $\mathcal{X} = \mathcal{TREE}$, then the root of the extracted tree is a correct process.*
- *If $\mathcal{X} \in \{\mathcal{SC}, \mathcal{COMPLETE}, \mathcal{RING}, \mathcal{BIC}\}$, then the extraction is exact.*

Proof. For $\mathcal{STAR}$ and $\mathcal{TREE}$, all the dicut reductions of the extracted graph contain at least respectively the center and the root, then the restriction of the extracted graph contains at least these nodes, proving that they are correct processes.

There is no dicut for a strongly connected graph. Hence in $\mathcal{SC}$, there is no dicut reduction then by the Closure property the subgraph induced by the set of correct processes of the extracted graph is the extracted graph itself. $\mathcal{COMPLETE}$, $\mathcal{RING}$, and $\mathcal{BIC}$ are particular cases of systems only composed of strongly connected timeliness graphs.

An immediate consequence of Proposition 2 is that any extraction algorithm gives an implementation of eventual leader election (failure detector Ω) for systems $\mathcal{STAR}$ and $\mathcal{TREE}$ as well as an implementation of failure detector $\Diamond\mathcal{P}$ for systems $\mathcal{COMPLETE}$, $\mathcal{RING}$, $\mathcal{SC}$ and $\mathcal{BIC}$.

Due to the lack of space, the proofs of the two following propositions have been omitted. In the first proposition we show that extraction is not always possible. Actually, in the proof we exhibit some non dicut-closed systems, namely $\mathcal{PAIR}$, where no extraction algorithm can be implemented.

Proposition 3. *There exist some systems $\mathcal{X}$ for which there is no extraction algorithm.*

In the next section we show that for all dicut-closed systems there is an extraction algorithm. For systems like $\mathcal{STAR}$, $\mathcal{TREE}$ and $\mathcal{PAIR}$, there exists no *exact* extraction algorithm.

Proposition 4. *There exist some systems $\mathcal{X}$ for which there is an extraction algorithm and there is no exact extraction algorithm.*

4 An Extraction Algorithm

The aim of this section is to show that the dicut-closed property of a system is sufficient to solve the extraction problem. To that end, we propose in Figure 1 an extraction algorithm, called $\mathcal{A}(\mathcal{X})$, for dicut-closed systems $\mathcal{X}$.

The basic idea of Algorithm $\mathcal{A}(\mathcal{X})$ is to make processes select a graph that is compatible with the timeliness graph of the run. For this, each process maintains for each graph x in $\mathcal{X}$ an *accusation counter* $Acc[x]$. This counter infinitely grows if some correct process is not in x or if some directed edge of x is not timely. Then, $Acc[x]$ is bounded if and only if x contains all correct processes and all timely links between pairs of correct processes.

We implement accusation counters as follows. A process regularly blames all the graphs in $\mathcal{X}$ in which it is not a node: it increments the accusation counters of all these graphs. Note that if the process is correct this accusation is justified and if the process is not correct, after some time, the process being dead stops to increment the accusation counters. Moreover, each process regularly sends on its outgoing links *alive* messages. Each process p maintains an estimate of the communication delays for each incoming link ($\Delta[q]$ for the incoming link (q, p)). If it does not receive *alive* messages within these estimates on some incoming link it blames all timeliness graphs in $\mathcal{X}$ containing this link (*i.e.*, increments the accusation counters for these graphs). As the estimate of the communication delay may be too short, each time it is exceeded the process increases it for the link. In this way, if the link is timely, at some time the estimate will be greater than the bound on communication delay.

The accusation counters are broadcast by reliable broadcasts. Each time a process receives a new value of accusation counter it updates its own accusation counter to the maximum of the received values and its current values. Hence, if some timely graph stops to be blamed then all correct processes eventually agree on the value of its accusation counter.

By selecting the graph G with the lowest accusation value (to break ties, we assume a total order among the graphs of $\mathcal{X}$) if any, correct processes eventually agree on the same timeliness graph of $\mathcal{X}$, moreover we can prove that this graph contains (1) all the correct processes, and (2) all edges between correct processes are timely links. As a consequence, the Convergence, the Compatibility and the Validity properties of the extraction algorithm are ensured. Nevertheless, this graph can also contain faulty processes and edges between correct and faulty processes.

Consider now the Closure property. If G contains only correct processes then the Closure property is trivially satisfied. Otherwise, G contains $Correct(r)$ and a set F of faulty processes. In this case, $(Correct(r), F)$ is a dicut reduction of G: Indeed if there is an edge in G from a faulty process q to a correct process p, eventually the process p stops to receive messages from q and the accusation counter of G grows infinitively often. Hence, in all cases, the Closure property is satisfied.

Hence, if $\mathcal{X}$ is dicut-closed, Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$. Moreover from Proposition 2, if all the graphs of $\mathcal{X}$ are strongly connected then the algorithm exactly extracts a graph in $\mathcal{X}$.

In the algorithm, each process p uses local timers, one per process. The timer of p dedicated to q is set (by setting `settimer`(q) to a positive value) to a time interval rather than absolute time. The timer is decremented until it expires. When the timer expires `timerexpire`(q) becomes *true*. Note that a timer can be restarted before it expires.

In the algorithm, we denote by $\prec$ the total order relation on $\mathcal{X}$ and by $\prec_{lex}$ (see Line 2) the total order relation defined as follows: $\forall x, y \in \mathcal{X}$, $\forall c_x, c_y \in \mathbb{N}$, $(c_x, x) \prec_{lex} (c_y, y) \equiv [c_x < c_y \vee (c_x = c_y \wedge x \prec y)]$.

```
Code for each process $p$
 1: Procedure $updateExtractedGraph()$
 2:     $G \leftarrow x$ such that $(Acc[x], x) = \min_{\prec_{lex}}\{(Acc[x'], x')$ such that $x' \in \mathcal{X}\}$

 3: On initialization:
 4: for all $x \in \mathcal{X}$ do $Acc[x] \leftarrow 0$
 5: for all $q \in \Pi \setminus \{p\}$ do
 6:     $\Delta[q] \leftarrow 1$
 7:     settimer$(q) \leftarrow \Delta[q]$
 8: $updateExtractedGraph()$
 9: start tasks 1 and 2

10: task 1:
11:     loop forever
12:         send$\langle alive \rangle$ to every $q \in \Pi \setminus \{p\}$ every $K$ time
13:         rbroadcast$\langle ACC, \perp, p \rangle$ every $K$ time  /* to accuse graphs that do not contain $p$ */

14: task 2:
15:     upon receive$\langle alive \rangle$ from $q$ do
16:         settimer$(q) \leftarrow \Delta[q]$

17:     upon timerexpire$(q)$ do
18:         rbroadcast$\langle ACC, q, p \rangle$  /* to accuse graphs that contain the link $(q, p)$ */
19:             $\Delta[q] \leftarrow \Delta[q] + 1$
20:             settimer$(q) \leftarrow \Delta[q]$

21:     upon rdeliver$\langle ACC, q, h \rangle$ do /* information from $h$ */
22:         for all $x \in \mathcal{X}$ do
23:             if $q = \perp$ then
24:                 if $h \notin Node(x)$ then $Acc[x] \leftarrow Acc[x] + 1$
25:             else
26:                 if $(q, h) \in Edge(x)$ then $Acc[x] \leftarrow Acc[x] + 1$
27:         $updateExtractedGraph()$
```

Fig. 1. Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$

A sketch of the correctness proof of $\mathcal{A}(\mathcal{X})$ is given below. In this sketch, we consider a run r of $\mathcal{A}(\mathcal{X})$ in dicut-closed system $\mathcal{X}$. We will denote by var_p^t the value of var of process p at time t.

We first notice that all variables $Acc_p[x]$ are monotonically increasing:

Lemma 2. *For all times t and t' such that $t \geq t'$, for all processes p, for all graphs x in $\mathcal{X}$, $Acc_p^t[x] \geq Acc_p^{t'}[x]$.*

Let $\sup(Acc_p[x])$ be the supremum of $Acc_p^t[x]$ for all t, we say that $Acc_p[x]$ is unbounded if $\sup(Acc_p[x])$ is equal to ∞ and bounded otherwise. As $Acc_p[x]$ is also updated by reliable broadcast each time some process q modifies $Acc_q[x]$ we have:

Lemma 3. *For all correct processes p and q, for all graphs x in $\mathcal{X}$,* $\sup(Acc_p[x]) = \sup(Acc_q[x])$

Let $\sup(Acc[x])$ be the supremum $\sup(Acc_p[x])$ over all correct processes p of $Acc_p[x]$ (by Lemma 3, $\sup(Acc[x])$ is well-defined). If there is a least one $x \in \mathcal{X}$ such that $\sup(Acc[x])$ is bounded, then $\min\{\sup(Acc[x'])|x' \in \mathcal{X}\}$ is finite, hence G the graph such that $(Acc[G], G) = \min_{\prec_{lex}}\{(Acc[x'], x')|x' \in \mathcal{X}\}$ is well defined. Then all correct processes converge to the same graph:

Lemma 4. *If there exists x in $\mathcal{X}$ such that* $\sup(Acc[x])$ *is bounded then there is a time after which for every correct process p, G_p is G.*

Now we prove the Compatibility property. Consider any timeliness graph $x \in \mathcal{X}$ compatible with $T(r)$. Then there is a time t after which all faulty processes are dead and the estimates of communication delays are greater than the bounds of communication delays of timely links of the run. After time t, (1) as x contains all correct processes, no process will blame x because it is not a node of x, and (2) as all edges of x are timely, no process will blame x for one of its edges then:

Lemma 5. *If x in $\mathcal{X}$ is compatible with $T(r)$, then* $\sup(Acc[x])$ *is bounded.*

Reciprocally, let x be a timeliness graph of $\mathcal{X}$ that is not compatible with the run. If process p is correct and p is not in x, it regularly blames x then $\sup(Acc[x]) = \infty$. If process p is not correct there is a time t after which it does not send any *alive* message, and there is a time after the timers on p expire forever for all correct processes, then if p is in x, $Acc_p[x]$ is incremented infinitely often and $\sup(Acc[x]) = \infty$. In the same way if q is correct and (p, q) is not timely, by the fifo property of the link, the timer for p expires infinitely often for process q and if (p, q) is an edge of x then $Acc_q[x]$ is incremented infinitely often and $\sup(Acc[x]) = \infty$.

Then:

Lemma 6. *For every x in $\mathcal{X}$, if* $\sup(Acc[x])$ *is bounded then $x[Correct(r)]$ is compatible with $T(r)$.*

Lemma 4 and Lemma 5 prove the Convergence property. Let G be the timeliness graph such that for every correct process p eventually $G_p = G$. Hence by Lemma 6:

Lemma 7. *$G[Correct(r)]$ is compatible with $T(r)$.*

It remains to prove that G satisfies the Closure property: $G[\mathit{Correct}(r)]$ is a dicut reduction of G or is equal to G. As $G[\mathit{Correct}(r)]$ is compatible with $T(r)$, we have:

Lemma 8. *$\mathit{Correct}(r) \subseteq \mathit{Node}(G)$.*

Let $F = \mathit{Node}(G) - \mathit{Correct}(r)$. If F is empty the Closure property is trivially ensured. Consider now the case where F is not empty. F contains only faulty processes and $(\mathit{Correct}(r), F)$ is a partition of $G(\mathit{Node})$. If there is an edge in $\mathit{Edge}(G)$ from a faulty process q to a correct process p, eventually the process p never receives a message from q and the accusation counter of G will be unbounded, contradicting the choice of G. So, we have:

Lemma 9. *If $F \neq \emptyset$ then $\mathit{Edge}(G) \cap (F \times \mathit{Correct}(r)) = \emptyset$.*

Hence, $(\mathit{Correct}(r), F)$ is a dicut of G.

Lemma 4 and Lemma 5 prove the Convergence property, Lemma 7 proves the Compatibility property and Lemma 9 proves the Closure property. Moreover, G is clearly in $\mathcal{X}$ proving the Validity. Proposition 2 shows that the extraction is exact when all graphs of $\mathcal{X}$ are strongly connected. Hence, we can conclude with the following theorem:

Theorem 1. *Let $\mathcal{X}$ be a dicut-closed system. Algorithm $\mathcal{A}(\mathcal{X})$ extracts a graph in $\mathcal{X}$. Moreover if all graphs of $\mathcal{X}$ are strongly connected, Algorithm $\mathcal{A}(\mathcal{X})$ exactly extracts a graph in $\mathcal{X}$.*

5 An Efficient Extraction Algorithm

In this section, we propose another extraction algorithm called $\mathcal{AF}(\mathcal{X})$ (Figures 2 and 3). This algorithm is efficient meaning that the (correct) processes eventually only send messages along the edges of the extracted graph.

$\mathcal{AF}(\mathcal{X})$ (exactly) extracts a timeliness graph from system $\mathcal{X}$, where (1) $\mathcal{X}$ is dicut-closed and (2) for all graphs $g \in \mathcal{X}$ there is some process p, called *root*, such that there is a directed path from p to every node of g. For example, $\mathcal{TREE}$ and $\mathcal{RING}$ systems have this property.

In the following, we refer to these systems as *dicut-closed systems with a root.* For every graph g in $\mathcal{X}$, the function $\mathit{root}(g)$ returns a root of g.

In the algorithm, every process p stores several values concerning the graphs $x \in \mathcal{X}$ such that $\mathit{root}(x) = p$: (1) $\mathit{Acc}[x]$ is the accusation counter of x whose goal is the same as in Algorithm 1, (2) $\mathit{Prop}[x]$ is a *proposition counter* whose goal will be explained later, and (3) $\Delta[x]$ gives the expected time for a message to go from p (the root of the x) to all the nodes of x.

Every process also maintains a set variable $\mathit{Candidates}$. Each element of this set is a 4-tuple composed of a graph x of $\mathcal{X}$ and the newest values of $\mathit{Acc}[x]$, $\mathit{Prop}[x]$, and $\Delta[x]$ known by the process (the exact values are maintained at $\mathit{root}(x)$). Each element in this set is called *candidate* and each process selects its extracted graph among the graphs in the candidate elements.

As in Algorithm 1:

(1) Each process p sends *alive* messages on its outgoing links and monitors its incoming links. However, we restrain here the *alive* message sendings: process p sends *alive* messages on its outgoing link (p, q) only if (p, q) is in a graph candidate.

(2) A graph candidate is blamed if (a) a correct process is not in the graph or (b) a process receives an out of date message through one of its incoming links. In both cases the candidate is definitely removed from the $Candidates$ sets of all processes. To achieve this goal the process sends an accusation message (ACC) using a reliable broadcast and uses an array $Heard$ that ensures that an identical candidate (that is, the same graph with the same accusation and proposition values) can never be added again. Moreover, upon delivery of an accusation message for graph x, $root[x]$ increments $Acc[x]$.

We now present different mechanisms used to obtain the efficiency.

For all graphs $x \in \mathcal{X}$, only the process $root(x)$ is allowed to propose x as a candidate to the rest. Each process p stores its better candidate in its variable me, that is, the least blamed graph x such that $root(x) = p$.

- If a process finds in $Candidates$ a better candidate than me, it removes me from $Candidates$.
- If a process finds that me is better, it adds me to $Candidates$ and sends a *new* message containing me (1) to all processes that are not in $Node(me)$, and (2) to immediate successors of p in me. The immediate successors in me add me to their $Candidates$ set and relay the *new* message, and so on. By the reliability of the links, every correct process that is not in me eventually receives this message and blames me.

These mechanisms are achieved by the procedure *updateExtractedGraph*(). This procedure is called each time a graph candidate is blamed or a new candidate is proposed. Note that the $Candidates$ set is maintained with the set $OtherCand$ (the candidates of other processes), a boolean *Local* that is true when the process has a candidate, and me, the graph candidate.

A process p may give up a candidate without this candidate being blamed: in this case, p is the root of the candidate, it finds a better candidate in $OtherCand$, and removes me from $Candidates$. Then, p must not increment $Acc[me]$ when it receives accusations caused by this removing, indeed these accusations are not due to delayed messages. That is the goal of the proposition counter ($Prop$): in $Prop[x]$, $root(x)$ counts the number of times it proposes x as candidate and includes this value in each of its *new* messages (to inform other processes of the current value of the counter). Hence, when q wants to blame x, it now includes its own view of $Prop[x]$ in the accusation message. This accusation will be considered as legitimate by $root[x]$ (that is, will cause an increment of $Acc[x]$) only when the proposition counter inside the message matches $Prop[x]$. Also, whenever $root[x]$ removes x from $Candidates$, $root[x]$ increments $Prop[x]$ and does not send the new value to the other processes. In this way accusations due to this removing will be ignored.

For any timely candidate, the accusation counter will be bounded and its proposition counter increased each time it is proposed. In this way the graph with the smallest accusation and proposition values eventually remains forever in the *Candidates* set of all correct processes and it is chosen as extracted graph. (This is done in the procedure *updateExtractedGraph*().) Moreover, eventually all other candidates are given up and it remains only this graph in *Candidates*. In this way, only *alive* messages are sent and they are sent along the directed edges of the extracted graph ensuring the efficiency.

```
Code for each process $p$
Procedure $updateExtractedGraph()$
    Let $(a_{min}, min) = \min_{\prec_{lex}}\{(acc, c)$ such that $(c, acc, -, -) \in OtherCand\} \cup \{(\infty, \infty)\}$
    if $(a_{min}, min) < (Acc[me], me) \wedge Local$ then  /* Give up $me$ */
        rbroadcast$\langle ACC, me, Acc[me], Prop[me], \Delta[me]\rangle$
        $Prop[me] \leftarrow Prop[me] + 1$
        $Local \leftarrow false$
    $Candidates \leftarrow OtherCand$
    $me \leftarrow x$ such that $(a, x) = \min_{\prec_{lex}}\{(acc, c)$ such that $c \in \mathcal{X} \wedge root(c) = p\}$
    if $(Acc[me], me) < (a_{min}, min) \wedge Local = false$ then  /* Propose $me$ */
        $Local \leftarrow true$
        $Candidates \leftarrow Candidates \cup \{(me, Acc[me], Prop[me], \Delta[me])\}$
        send$\langle new, me, Acc[me], Prop[me], \Delta[me]\rangle$ to every process not in $Node(me)$
        for all $h \in \Pi \setminus \{p\}$ do
            if $(h,p) \in Edge(me)$ then
                $\Delta[h] \leftarrow \max(\Delta[h], \Delta[me])$
                settimer$(h) \leftarrow \Delta[h]$
            if $(p,h) \in Edge(me)$ and $h \neq root(me)$ then
                send$\langle new, me, Acc[me], Prop[me], \Delta[me]\rangle$ to $h$
    $G \leftarrow x$ such that $(a, x) \min_{\prec_{lex}}\{(a', x')$ such that $(x', a', p', d') \in Candidates\}$
```

Fig. 2. Procedure updateExtractedGraph of Algorithm $\mathcal{AF}(\mathcal{X})$

The following theorem states the correctness of $\mathcal{AF}(\mathcal{X})$. For space consideration, its proof has been omitted.

Theorem 2. *Let $\mathcal{X}$ be a dicut-closed system with a root. Algorithm $\mathcal{AF}(\mathcal{X})$ efficiently extracts a graph in $\mathcal{X}$. Moreover if all graphs of $\mathcal{X}$ are strongly connected, Algorithm $\mathcal{AF}(\mathcal{X})$ efficiently and exactly extracts a graph in $\mathcal{X}$.*

6 Conclusion

Failure detector implementations in partially synchronous models generally use the timeliness properties of the system to approximate the set of correct (or faulty) processes. In some way, the extraction problem is a kind of generalization: instead of only searching the set of correct processes, here we try to extract also information about the timeliness of links. Besides, our solutions are based on already existing mechanisms used in failure detectors implementations as in [2, 3].

Information about the timeliness of links is useful for efficiency of fault-tolerant algorithms. In particular, in any extracted graph, any path between

```
Code for each process $p$
On initialization:
for all $x \in \mathcal{X}$ such that $root(x) = p$ do
    $Acc[x] \leftarrow 0$; $Prop[x] \leftarrow 0$; $\Delta[x] \leftarrow n$
for all $x \in \mathcal{X}$ such that $root(x) \neq p$ do $Heard[x] \leftarrow (-1, -1)$
for all $q \in \Pi \setminus \{p\}$ do $\Delta[q] \leftarrow 1$
$OtherCand \leftarrow \emptyset$
$Local \leftarrow false$
$me \leftarrow min\{x$ such that $x \in \mathcal{X} \wedge root(x) = p\}$
$updateExtractedGraph()$
start tasks 1 and 2

task 1:
    loop forever
        send$\langle alive \rangle$ to every process $q$ such that $\exists(x,\text{-},\text{-},\text{-}) \in Candidates$ and $(p, q) \in Edge(x)$ every $K$ time

task 2:
    upon receive$\langle alive \rangle$ from $q$ do
        settimer$(q) \leftarrow \Delta[q]$

    upon timerexpire$(q)$ do  /* Link $(q, p)$ is not timely, blame all candidates that contain $(q, p)$ */
        for all $(x, a, pr, d) \in OtherCand$ such that $(q, p) \in Edge(x)$ do
            rbroadcast$\langle ACC,x,a,pr,d \rangle$
        if $(q, p) \in Edge(me)$ then
            rbroadcast$\langle ACC,me,Acc[me],Prop[me],\Delta[me] \rangle$

    upon receive$\langle new, x, a, pr, d \rangle$ from $q$ do /* Proposition of a new candidate */
        if $p \notin Node(x)$ then  /* Blame $x$ that does not contain $p$ */
            rbroadcast$\langle ACC,x,a,pr \rangle$
        else
            $newCand \leftarrow false$
            if $(x, -, -, -) \notin OtherCand$ and $Heard(x) < (a, pr)$ then  /* New candidate */
                $newCand \leftarrow true$
            if $\exists(x, a_c, pr_c, d_c) \in OtherCand$ with $(a_c, pr_c) < (a, pr)$ then  /* New candidate */
                $OtherCand \leftarrow OtherCand \setminus (c, a_c, pr_c, d_c)$
                $newCand \leftarrow true$
            if $newCand$ then
                $OtherCand \leftarrow OtherCand \cup (x, a, pr, d)$
                $updateExtractedGraph()$
                $Heard[x] \leftarrow (a, pr)$
                for all $h \in \Pi \setminus \{p\}$ do
                    if $(h,p) \in Edge(x)$ then
                        $\Delta[h] \leftarrow \max(\Delta[h], d)$
                        settimer$(h) \leftarrow \Delta[h]$
                    if $(p,h) \in Edge(x)$ and $h \neq root(x)$ then send$\langle new, x, a, pr, d \rangle$ to $h$

    upon rdeliver$\langle ACC,x,a,pr,d \rangle$ do
        if $root(x) = p$ then
            if $x = me \wedge a = Acc[me] \wedge pr = Prop[me]$ then  /* Check if the accusation is up to date */
                $Acc[me] \leftarrow Acc[me] + 1$; $\Delta[me] \leftarrow \Delta[me] + 1$
                $Local \leftarrow false$
        else
            $OtherCand \leftarrow OtherCand \setminus (x, a, pr, d)$
            if $Heard[x] < (a, pr)$ then $Heard[x] \leftarrow (a, pr)$
        $updateExtractedGraph()$
```

Fig. 3. Algorithm $\mathcal{AF}(\mathcal{X})$ that efficiently extracts a graph in $\mathcal{X}$

a pair of correct processes is only constituted of timely links. This property is particulary interesting to get efficient routing algorithms.

We gave an extraction algorithm for dicut-closed sets of timeliness graphs. Moreover, we proved that the extraction is exact when all the timeliness graphs are also strongly connected.

Given dicut-closed timeliness graphs that contain a root, we shown how to efficiently extract a graph from it. By efficiency we mean giving a solution where eventually messages are only sent over the links of the extracted graph.

It is important to note that the main purpose of the algorithms we proposed is to show the feasability of the extraction under some conditions. So, the complexity of our algorithms was not the main focus of this paper.

As a consequence, our algorithms are somehow unrealistic because of their high complexity. Giving more practical solutions will be the purpose of our future works.

Acknowledgments. We are grateful to members of the *Graph* team of the *LIAFA* Lab for the helpful discussions and their interesting suggestions.

References

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Stable leader election. In: Welch, J.L. (ed.) DISC. Lecture Notes in Computer Science, vol. 2180, pp. 108–122. Springer (2001)
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega with weak reliability and synchrony assumptions. In: PODC. pp. 306–314 (2003)
3. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Communication-efficient leader election and consensus with limited link synchrony. In: Chaudhuri, S., Kutten, S. (eds.) PODC. pp. 328–337. ACM (2004)
4. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega in systems with weak reliability and synchrony assumptions. Distributed Computing 21(4), 285–314 (2008)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
6. Dolev, D., Dwork, C., Stockmeyer, L.J.: On the minimal synchronism needed for distributed consensus. Journal of the ACM 34(1), 77–97 (1987)
7. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
8. Delporte Gallet, C., Devismes, S., Fauconnier, H., Larrea, M.: Algorithms For Extracting Timeliness Graphs, http://hal.archives-ouvertes.fr/hal-00454388/en/
9. Hadzilacos, V., Toueg, S.: A modular approach to fault-tolerant broadcasts and related problems. Tech. Rep. TR 94-1425, Department of Computer Science, Cornell University (1994)
10. Hutle, M., Malkhi, D., Schmid, U., Zhou, L.: Chasing the weakest system model for implementing omega and consensus. IEEE Trans. Dependable Sec. Comput. 6(4), 269–281 (2009)
11. Larrea, M., Arévalo, S., Fernández, A.: Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In: Jayanti, P. (ed.) DISC. Lecture Notes in Computer Science, vol. 1693, pp. 34–48. Springer (1999)
12. Mostéfaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: DSN. pp. 351–360. IEEE Computer Society (2003)