Approximation of δ-Timeliness[⋆]

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

[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

Abstract. In asynchronous message-passing distributed systems prone to process crashes, a communication link is said δ-*timely* if the communication delay on this link is bounded by some constant δ. We study here in which way processes may approximate and find structural properties based on δ-timeliness (*e.g.*, find δ-timely paths between processes or build a ring between correct processes using only δ-timely links).
To that end, we define a notion of approximation of predicates. Then, with help of such approximations, we give a general algorithm that enables to choose and eventually agree on one of these predicates. Finally, applying this approach to δ-timeliness, we give conditions and algorithms to approximate δ-timeliness and dynamically find structural properties using δ-timeliness.

1 Introduction

Assume an asynchronous message-passing system prone to process crash failures and consider the following problem: we know that after some unknown time there is at least one path from process p to process q such that every message sent along this path arrives to process q by δ time units (such a path is said δ-timely). Now, how can we determine one of these paths or at least one path that is Δ-timely for a Δ close to δ? By "determine" we mean that eventually all processes agree on the chosen path.

To that end, the processes must be at least able to test the δ-timeliness of paths and one of the contribution of this paper is to give some necessary and sufficient conditions to do this. In particular, we prove that without synchronized clocks, the system has to ensure strong synchrony properties in the sense that there must not only exist δ-timely paths from p to q but also from q to p.

The δ-timeliness of a path is a property that is only eventually ensured. Moreover, δ-timeliness of a link can only be approximated by processes. We would like that an approximation algorithm outputs boolean values in such a way that if the path is truly δ-timely, then eventually only true is output, and if the path is not δ-timely, then false is output infinitely often. However as we will see, such approximations of δ-timeliness are generally too strong because there is some incertitude on the time it takes to go from p to q on the path. Therefore the approximation algorithm only ensures that if the path is δ-timely, then true is eventually output forever and if the path is not Δ-timely (with $\Delta > \delta$), then false is output infinitely often. Hence, between δ and Δ the approximation algorithm may output true or false just as well.

The existence of such approximations enables to answer to our initial problem (but with the little restriction on δ and Δ): if at least one δ-timely path exists from p to q, it is then possible to ensure that all processes eventually agree on the same Δ-timely

[⋆] This work has been supported in part by the ANR project *SHAMAN* and the INRIA project *GANG*.

path from p to q. An algorithm which ensures that all correct processes eventually agree on the same structure verifying some properties is called an *extraction* algorithm [7]. Many other problems can be solved in the same way: for example extraction of a tree containing all correct processes whose all paths from the root are Δ-timely, or a ring containing all correct processes and whose links are Δ-timely, etc.

Actually, all these aforementioned algorithms use methods very similar to the ones used, for example, in the eventual leader election problem in [1–4, 9, 11]. In fact, we prove here that this approach can be expressed in a very general framework. Approximation of δ-timeliness and extraction of some structures based on the δ-timeliness relation are special cases of the more general problem of approximating properties on runs and extracting structures based on these properties. Instead of δ-timeliness, one can consider more general properties on runs and consider when they are eventually true forever. Then, assuming approximation algorithms for these predicates, it is possible to extract (*i. e.* choose and agree) one of these predicates that is true in the run. More precisely, as for δ-timeliness, the approximation is defined by pairs of predicates (P, Q), where P specifies when the approximation has to eventually output true forever and $\neg Q$ when the approximation has not to eventually output true forever (in other words has to output infinitely often false). Then, given a set of pairs of predicates (P_i, Q_i) indexed by some set I, the extraction algorithm, assuming that at least one P_i is true, will converge to some i_0 for all correct processes such that Q_{i_0} is true. This generalization enables to have the same algorithms for many different problems.

In this way, the extraction of structures based on the δ-timeliness relation is only a special case of this general case. For example, assuming that processes have a trusting mechanism (like a failure detector) giving to each processes lists of process supposed to be alive, and consider the predicate "p is eventually in all lists", then the extraction algorithm gives one of such processes. Assuming that these lists are given by failure detector $\Diamond\mathcal{S}$ ([5]) then the extraction algorithm gives an implementation of failure detector Ω and the algorithm is rather close to algorithm of [6].

Contributions. In this paper, we first define a general framework in which processes may approximate predicates on runs and give a generic extraction algorithm that enables processes to converge on one of the predicates satisfied in the run. Then, we apply these concepts by proposing algorithms and impossibility results about the approximation of δ-timeliness. In particular we give sufficient conditions to approximate δ-timeliness on links. More precisely, we prove that we need either to have perfectly synchronized clocks or to assume very strong timeliness requirements. Finally, we give examples of general extractions based on approximation of link δ-timeliness. These illustrations emphasizes the two mains points of our contribution. Firstly, this general approach allows to drastically simplify the design of algorithm. Indeed, from a simple algorithm that approximates a local predicate, like "a link is δ-timely", we can easily derive an algorithm to extract a more complex structure such as a δ-timely path or a δ-timely ring. Secondly, our algorithms have practical applications, as they can be used to find efficient routes in a network (that is, δ-timely paths), or efficient overlay such as δ-timely rings.

Related works. Many works [1–4, 9, 11] on eventual leader election or Ω implementations uses the same techniques as here. This paper may be also seen as a formalization and abstraction of these techniques. To the best of our knowledge [7] was the first one to study timeliness for itself and not as a mean to get information about process crashes. Note that as link timeliness only means that the communication delay of the link is bounded, its interest is mainly theoretical.

Roadmap. In Section 2 we present the model. Section 3 gives the basic definitions for approximations of predicates and the general extraction algorithm. Section 4 gives conditions for the existence algorithms for approximation of δ-timeliness. In Section 5 we give some examples of applications of extraction algorithms for δ-timeliness.

2 Model

Processes. We consider distributed systems composed of n processes that 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, denoted by (p, 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.

We assume the existence of a discrete global clock to which processes cannot access. The range $\mathcal{T}$ of the clock's ticks is the set of natural integers. All correct processes p are *synchronous*, *i.e.*, they are able to take step at each clock tick. So, they can accurately measure time intervals.

As processes can accurately measure time intervals, our algorithms can use *local timers*. A process p starts a timer by setting `settimer`(F) to a positive value. F is a flag that identifies the timer. The timer F is then decremented until it expires. When the timer expires, `timerexpire`(F) becomes $true$. Note that a timer can be restarted (by setting `settimer`(F) to some positive value) before it expires. Finally, `unsettimer`(F) allows to disable timer F.

Communication pattern. A *communication pattern* CP is a function from $\mathcal{T} \times \Pi \times \Pi$ to $\mathbb{N}$: $CP(\tau, p, q) = k$ means that if process p sends a message to process q at time τ then the message is received by process q at time $\tau + k$.

Communication between processes is assumed to be *reliable* and *FIFO*. *Reliable* means that (1) every message sent through a link (p, q) is eventually received by q if q is correct, also (2) if a message m from p is received by q, then m is received by q at most once and only if p previously sent m to q. *FIFO* means that messages from p to q are received in the order they are sent. The links being reliable, an implementation of *reliable broadcast* [8] is possible and in the following reliable broadcast is defined by two primitives: `rbroadcast`$\langle\rangle$ and `rdeliver`$\langle\rangle$.

Runs. An algorithm $\mathcal{A}$ consists of n deterministic (infinite) automata, one per process. 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 $R = \langle CP, I, E, S, F\rangle$ where CP is a communication pattern, I is the initial state of the processes in Π, E is an infinite sequence of steps of $\mathcal{A}$, S is a list of increasing time values indicating when each step in E occurred, and F is a failure pattern, *i.e.* a non decreasing function from $\mathcal{T}$ to 2^{Π} such that $F(\tau)$ is the set of processes that are crashed at time τ. $Faulty(R) = \bigcup_{\tau\in\mathcal{T}} F(\tau)$ is the set of faulty processes and $Correct(R) = \Pi - Faulty(R)$ is the set of correct processes.

Process p may take a step at time τ only if it is not crashed at this time. A process p takes an infinite number of steps if and only if $p \in Correct(R)$. Moreover a run satisfies properties concerning sending and receiving messages according to CP: if p sends message m to q at time τ, and q is correct then m is received by q at time $\tau + CP(\tau, p, q)$.

δ-timeliness. Given some δ and two correct processes p and q, we say that *link* (p,q) *is δ-timely* if and only if there is a time τ such that for all time $\tau' \geq \tau$ we have $CP(\tau', p, q) \leq \delta$. By convention, if q is faulty then link (p,q) is δ-timely, and if p is faulty and q is correct, link (p,q) is not δ-timely.

3 Approximation and extraction

Predicates. Given a run R, the local state of process p in R at time τ is denoted $S_R(p,\tau)$. Predicates considered here are defined from functions ϕ from $\Pi \times \mathcal{T}$ to $\{true, false\}$ that define the truth value in local states $S_R(p,\tau)$. We always assume that $\phi(p,\tau) = true$ if p is crashed at time τ. By extension, $\phi(\tau)$ denotes $\bigwedge_{p\in\Pi} \phi(p,\tau)$. By definition, the predicate associated to ϕ, denoted P_ϕ, is true for run R if and only if there is a time τ_0 such that for all time $\tau \geq \tau_0$, $\phi(\tau)$ is true. In this case, we also say that ϕ is eventually forever true. In the following, a predicate P on run R is always associated in this way to some ϕ. When the context is clear we do not give ϕ explicitly.

For example let Q be the predicate "the boolean variable v of process p is eventually forever true". Q is a predicate on the run for which $\phi(q,\tau)$ is true if and only if v is true in $S_R(p,\tau)$. Remark that $\neg Q$ is true if and only if we have "the boolean variable v of process p is infinitely often false". Generally, $\neg P_\phi$ is equivalent to for all time τ there is some $\tau' \geq \tau$ and some process p such that $\phi(p,\tau')$ is false, that is, there is a process p such that $\phi(p,\tau)$ is false infinitely often.

Predicate P_ϕ is said to be *local to process* p if and only if for all time τ we have $\phi(\tau) = \phi(p,\tau)$. Let $\psi_{p,q}$ be the function defined by $\psi_{(p,q)}(r,\tau) = true$ if and only if $CP(\tau,p,q) \leq \delta$. $P_{\psi_{p,q}}$ is the predicate corresponding to the δ-timeliness of the link (p,q). In the following, $P_{\psi_{(p,q)}}$ will be abbreviated as $T_{pq}(\delta)$. With help of the assumption made on predicates for faulty processes, this predicate is local to process q. In the same way, predicate "p is eventually crashed", denoted $P_{p\ is\ crashed}$, is local to p.

Approximation algorithms. In the following, we are interested in algorithms that approximate some predicates in the sense that to approximate P we want to get some variable v such that: P is true if and only if v is eventually forever true. Actually, such approximations are too strong, so we consider here weaker approximations: we define the approximation by a pair of predicates (P,Q) such that (1) $P \Rightarrow Q$, (2) if P is true then v must be eventually forever true, and (3) if Q is false then v is not eventually forever true.

More precisely, consider pair (P,Q) of predicates such that $P \Rightarrow Q$, an *approximation algorithm* $\mathcal{A}$ for process p of predicates (P,Q) is an algorithm with a special boolean variable local to p $Out^p_{\mathcal{A}}$ (actually the output of $\mathcal{A}$ for p) written only by algorithm $\mathcal{A}$ such that:

- if P is true then $Out^p_{\mathcal{A}}$ is eventually forever true
- if Q is false then $Out^p_{\mathcal{A}}$ is not eventually forever true (*i.e.* is infinitely often false)

By convention, we assume that if p is correct then $Out^p_{\mathcal{A}}$ is written infinitely often (as processes are synchronous it is always possible). Not that if P is false but Q is true then $Out^p_{\mathcal{A}}$ may be eventually true forever or infinitely often false. In this way, for $Q \wedge \neg P$ there is no requirement on the output of $\mathcal{A}$.

By extension, predicates (P,Q) are local to process p if both P and Q are local to p. In the case of approximation of predicates (P,Q) local to p, an approximation algorithm can be implemented for every correct process:

Proposition 1. *If $\mathcal{A}$ is an approximation algorithm for p of predicates local to p (P, Q), then for every correct process q there exists an approximation algorithm for q of predicate (P, Q).*

Sketch of Proof. Let $\mathcal{A}$ be an approximation algorithm for p of predicates local to p (P, Q).

Algorithm given in Figure 1 implements an approximation algorithm $\mathcal{B}$ of (P, Q) for every correct process. In this algorithm, each time algorithm $\mathcal{A}$ modifies $Out^p_{\mathcal{A}}$, (1) $Out^p_{\mathcal{B}}$ is written and (2) a message is (reliably) broadcast to inform every process. When a process q delivers this message, it writes its output value $Out^q_{\mathcal{B}}$ with the new value. If this new value is false then $true$ is written again into $Out^q_{\mathcal{B}}$.

Assume that p is faulty. Then, by assumption about faulty processes and local predicates, P is $true$. Also, by definition of the algorithm, every correct process q only finitely delivers messages from p. As a consequence, $Out^q_{\mathcal{B}}$ is eventually forever true.

Assume that p is correct. First, by (1) $Out^p_{\mathcal{B}}$ is eventually forever $true$ if and only if $Out^p_{\mathcal{A}}$ is eventually forever $true$. Then, by (2), for every correct process $q \neq p$, $Out^q_{\mathcal{B}}$ is eventually forever $true$ if and only if $Out^p_{\mathcal{A}}$ is eventually forever $true$.

Hence, if $\mathcal{A}$ is an approximation algorithm for p of predicates local to p (P, Q), then $\mathcal{B}$ is an approximation algorithm for every correct process of predicate (P, Q). $\square$

```
In Code for $\mathcal{A}$ of $p$:
1:    whenever $\mathcal{A}$ writes $Out^p_{\mathcal{A}}$ with $b$    /* $b$ is a boolean value */
2:        $Out^p_{\mathcal{B}} := b$;rbroadcast$\langle(P, b)\rangle$
In Code for process $q \neq p$:
1:    whenever rdeliver$\langle$(P,b)$\rangle$
2:        $Out^q_{\mathcal{B}} := b$
3:        if $\neg b$ then $Out^q_{\mathcal{B}} := true$
```

Fig. 1. $\mathcal{B}$, approximation algorithm of (P, Q) for every correct process

The next proposition can be verified using an algorithm very similar to the one given in Figure 1.

Proposition 2. *If $\mathcal{A}_{P,Q}$ and $\mathcal{A}_{P',Q'}$ are approximation algorithms for correct process p of predicates (P, Q) and correct process q of predicates (P', Q'), respectively, then for all correct processes r, there are approximation algorithms for r of predicates $(P \wedge P', Q \wedge Q')$.*

In the following we are only interested in approximation algorithms for *all* correct processes. Then, by default, an approximation algorithm of (P, Q) means an approximation algorithm for all correct processes.

Given a finite set of indexes I and a set of predicates indexed by I say $(P_i, Q_i)_{i \in I}$, the set $(\mathcal{A}_i)_{i \in I}$ denotes the *set of approximation algorithms of* $(P_i, Q_i)_{i \in I}$, that is, for each $i \in I$, $\mathcal{A}_i$ is an approximation algorithm for (P_i, Q_i).

Extraction algorithms. Consider a set of predicates indexed by some finite set and assume that at least one of these predicates is true. We would like that all correct processes choose and converge to the same predicate satisfied in the run. To evaluate these predicates, processes use approximations algorithms as defined just before. Hence, we do not have sets of single predicates but sets of pair of predicates (P_i, Q_i) indexed by some set in which P_i specifies when the approximation outputs true forever and $\neg Q_i$ when the approximation does not output true forever. Hence, if at least one of the P_i is true, the extraction algorithm has to converge to one index j such that Q_j is true. In this way, from a property guaranteed in the run for at least one P_i, we find an index i_0 such that Q_{i_0} is true. Of course, if $Q_i = P_i$ then the chosen index verifies P_{i_0}.

More precisely, let $(P_i, Q_i)_{i \in I}$ be a set of predicates indexed by I, an *extraction algorithm* for $(P_i, Q_i)_{i \in I}$ is an algorithm such that in each run R where at least one P_i is true, all correct processes eventually choose the same index $i_0 \in I$ such that Q_{i_0} is true. In other words, each process p has a variable d_p and in each run R where $\bigvee_{i \in I} P_i = true$, there is an index $i_0 \in I$ satisfying the following two properties:

– **Eventual Agreement:** there is a time τ after which for all correct processes p $d_p = i_0$,
– **Validity:** Q_{i_0} is true.

Let I be a finite set of indexes, and $(P_i, Q_i)_{i \in I}$ a set of predicates indexed by I, if $(\mathcal{A}_i)_{i \in I}$ is a set of approximations of $(P_i, Q_i)_{i \in I}$, then the algorithm in Figure 2 is an extraction algorithm for $(P_i, Q_i)_{i \in I}$.

In this algorithm, each process p associates a (local) counter variable $Acc[i]$ to each variable $Out^p_{\mathcal{A}_i}$. Each time $Out^p_{\mathcal{A}_i}$ becomes $false$ at p, p increments $Acc[i]$. Moreover, each p regularly sends Acc to all other processes. Upon receiving a message containing the array acc, a process locally updates $Acc[i]$ for all i with the maximum value between $Acc[i]$ and $acc[i]$. This way:

– If there is a time after which $Out^p_{\mathcal{A}_i}$ is true forever for all processes, then the value of $Acc[i]$ is eventually fixed to the same value for all correct processes.
– If $Out^q_{\mathcal{A}_i}$ is false infinitely often at some process q, then $Acc[i]$ is incremented infinitely often by q. Consequently, $Acc[i]$ is unbounded for all correct processes.

If for some j, P_j is true in the run then, as $\mathcal{A}_j$ approximates (P_j, Q_j), at least $Acc[j]$ is eventually fixed to the same value for all correct processes. To agree on some i, the processes call $updateExtracted()$ each time they modify their array: this function sets the variable d to the index i_0 such that $Acc[i_0]$ is minimum (we use the order on the indices to break tie). Hence, if for some j, P_j is true in the run then, eventually the d-variable of every correct process p is set to the same index i_0 such that $Out^p_{\mathcal{A}_{i_0}}$ is eventually forever true. As $\mathcal{A}_{i_0}$ approximates (P_{i_0}, Q_{i_0}), Q_{i_0} is true in the run and we can conclude:

Proposition 3. *Let I be a finite set of indexes, and $(P_i, Q_i)_{i \in I}$ a set of predicates indexed by I, if $(\mathcal{A}_i)_{i \in I}$ is a set of approximation algorithms of $(P_i, Q_i)_{i \in I}$, then there exists an extraction algorithm for $(P_i, Q_i)_{i \in I}$.*

Note that this extraction algorithm has two additional properties: it is self-stabilizing and may tolerates fair lossy links.

Unfortunately in this extraction algorithm all correct processes send infinitely many messages and consult infinitely many times all approximation algorithms $(\mathcal{A}_i)$. Now, it is possible to achieve a more efficient extraction concerning communication.

Let $(\mathcal{A}_i)_{i \in I}$ be a set of approximation algorithms of $(P_i, Q_i)_{i \in I}$. The extraction algorithm $\mathcal{A}$ obtained with $(\mathcal{A}_i)_{i \in I}$ is *communication-efficient* [10] if: (1) $\mathcal{A}$ is an extraction algorithm for $(P_i, Q_i)_{i \in I}$, and (2) for each run R if at least one P_i of $(P_i, Q_i)_{i \in I}$ is true then there is a time τ after which (a) there exists some j in I such that every correct process p reads only $Out^p_{\mathcal{A}_j}$, and (b) no message is sent by $\mathcal{A}$ after τ.

If $(\mathcal{A}_i)_{i \in I}$ is a set of approximations of $(P_i, Q_i)_{i \in I}$, then the algorithm in Figure 3 is an efficient extraction algorithm for $(P_i, Q_i)_{i \in I}$.

```
Code for each process $p$
1: Procedure $updateExtracted()$
2:     $d \leftarrow i$ such that $(Acc[i], i) = \min_{\prec_{lex}}\{(Acc[i'], i')$ such that $i' \in I\}$

3: On initialization:
4:     for all $i \in I$ do $Acc[i] \leftarrow 0$
5:     $updateExtracted()$
6:     start tasks 1, 2 and 3

7: task 1:
8:     loop forever
9:         each time $Out^p_{\mathcal{A}_i}$ becomes $false$
10:            $Acc[i] \leftarrow Acc[i] + 1$
11:            $updateExtracted()$

12: task 2:
13:     loop forever
14:         send$\langle(ACC, Acc)\rangle$ to every process except $p$ every $\eta$        $/* \eta$ is a constant $*/$

15: task 3:
16:     upon receive$\langle(ACC, a)\rangle$ do
17:         for all $i \in I$ do
18:             $Acc[i] \leftarrow max(Acc[i], a[i])$
19:         $updateExtracted()$
```

Fig. 2. Extraction algorithm for $(P_i, Q_i)_{i\in I}$ assuming $(\mathcal{A}_i)_{i\in I}$ is a set of approximations of $(P_i, Q_i)_{i\in I}$

Again in this algorithm, a counter variable $Acc[i]$ is associated to each variable $Out^p_{\mathcal{A}_i}$. Again, $updateExtracted()$ returns the index i_0 such that $Acc[i_0]$ is minimum and the variable d is set to this index. However, to get the efficiency, each process p now only tests the value of $Out^p_{\mathcal{A}_d}$. Each time $Out^p_{\mathcal{A}_d}$ becomes $false$, process p blames d by reliably broadcasting the message (ACC, d) to every process. Upon delivering (ACC, x), a process increments $Acc[x]$ and calls $updateExtracted()$ to refresh the value of d.

If for some i, P_i is true in the run then, as $\mathcal{A}_i$ approximates (P_i, Q_i), $Out^q_{\mathcal{A}_i}$ is eventually forever true at all correct processes p and the message (ACC, i) can only be finitely broadcasted. Moreover, by the property of the reliable broadcast, all correct processes delivers the same number of (ACC, i) messages. Hence, eventually every correct process agrees on a fixed value of $Acc[i]$. As the value in Acc are monotically increasing, by definitions of $updateExtracted()$ and the reliable broadcast, the d-variables of all correct processes eventually converge to the same index i_0.

Assume now that the d-variables of all correct processes eventually converge to the same index i_0 but $Out^q_{\mathcal{A}_{i_0}}$ is $false$ infinitely often for some correct process q. In this case, q continuously tests $Out^q_{\mathcal{A}_{i_0}}$ and, consequently, (reliably) broadcasts infinitely many (ACC, i_0) messages. So, the value of $Acc[i_0]$ grows infinitely often at every correct process and eventually the d-variables of all correct processes are set to some other index.

Hence, if for some j, P_j is true in the run then, eventually the d-variable of every correct process p is set to the same index i_0 such that $Out^p_{\mathcal{A}_{i_0}}$ is eventually forever true. As $\mathcal{A}_{i_0}$ approximates (P_{i_0}, Q_{i_0}), Q_{i_0} is true and we can conclude:

Proposition 4. *If $(\mathcal{A}_i)_{i\in I}$ is a set of approximation algorithms of $(P_i, Q_i)_{i\in I}$, then there exists an communication-efficient extraction algorithm for $(P_i, Q_i)_{i\in I}$.*

4 Approximation algorithms for δ-timeliness

We now consider the predicates $T_{qp}(\delta)$ on δ-timeliness of links (q, p). We notice that even in the good case where processes are equipped with perfectly synchronized clocks,

```
Code for each process $p$
 1: Procedure $updateExtracted()$
 2:     $d \leftarrow i$ such that $(Acc[i], i) = \min_{\prec_{lex}}\{(Acc[i'], i')$ such that $i' \in I\}$

 3: On initialization:
 4:     for all $i \in I$ do $Acc[i] \leftarrow 0$
 5:     $updateExtracted()$
 6:     start tasks 1 and 2

 7: task 1:
 8:     loop forever
 9:         each time $Out^p_{\mathcal{A}_d}$ becomes $false$
10:             rbroadcast$\langle(ACC, d)\rangle$

11: task 2:
12:     upon rdeliver$\langle(ACC,x)\rangle$ do
13:         $Acc[x] \leftarrow Acc[x] + 1$
14:         $updateExtracted()$
```

Fig. 3. Communication-efficient extraction algorithm for $(P_i, Q_i)_{i\in I}$ assuming $(\mathcal{A}_i)_{i\in I}$ is a set of approximations of $(P_i, Q_i)_{i\in I}$

process q has to send a message every tick of time to approximate $(T_{qp}(\delta), T_{qp}(\delta))$. This does not seem reasonable, so we consider the predicate $(T_{qp}(\delta), T_{qp}(\Delta))$ with $\Delta > \delta$. Obviously, we have a good approximation when Δ is close to δ. That is, if there exist two reasonable constants m and a such that $\Delta = m\delta + a$.

We first show that without additional assumptions there is no approximation algorithm for $(T_{qp}(\delta), T_{qp}(\Delta))$. Then we consider two assumptions that make the problem solvable: (1) each process is equipped with a perfectly synchronized clock, and (2) there is a Γ-timely path from q to p. We show that in both cases, there exist two constants m and a such that if $\Delta \geq m\delta + a$, then there is an algorithm to approximate $(T_{qp}(\delta), T_{qp}(\Delta))$.

4.1 Impossibility results

If the system does not have additional assumptions, like perfectly synchronized clocks in processes p and q or a Γ-timely path from q to p, then there is no algorithm to approximate $(T_{qp}(\delta), T_{qp}(\Delta))$. This result holds even if we consider a system without crash failures.

Proposition 5. *There is no approximation algorithm for* $(T_{qp}(\delta), T_{qp}(\Delta))$.

Sketch of Proof. In a run, it is possible that p starts executing its code at some time τ_0 while q starts at some time τ'_0. (If we know that p and q start executing their local code at the same time, then as their local clock can accurately measure the time, they have a perfect synchronized clock.)

We proceed by contradiction. Assume there is an approximation algorithm $\mathcal{A}$ for $(T_{qp}(\delta), T_{qp}(\Delta))$. Let R be a run of $\mathcal{A}$ in which (i) the link q to p is δ-timely, (ii) all messages from any process different from q take a time $K > \Delta + 2$, and (iii) all messages to any process different from p take a time $K > \Delta + 2$. R defines the real time at which processes takes steps, but by hypothesis processes do not have access to this time. It is then possible to construct a run R_K of $\mathcal{A}$ in which (1) process q takes the same steps at the same time than in R, and (2) for each other process r, r takes at time $\tau + K - 1$ the step it takes at time τ in R. For every process, R and R_K are indistinguishable. Now, the properties of the approximation algorithm gives that there is a time after which $Out_{\mathcal{A}}$ is forever true in R, and consequently, in R_K. However, in R_K, the messages sent by q are received by p with a delay of $K - 1 > \Delta$. That is, the communication from q to p is not Δ-timely in R_K, contradicting the properties of the approximation algorithm. □

Note that when it is possible to design, an approximation algorithm for $(T_{qp}(\delta), T_{qp}(\delta))$ is really expensive. To see this, assume that q sends a message at time $2\tau - 1$ and at time $2\tau + 1$. These messages are received by process p at $2\tau - 1 + \delta$ and $2\tau + 1 + \delta$. As q has omitted to send a message at time 2τ, we cannot evaluate if $CT(2\tau, q, p) = \delta$ or $\delta + 1$. In this latter case, the link is not δ-timely but p cannot observe that. Hence, follows:

Proposition 6. *When an approximation algorithm for $(T_{qp}(\delta), T_{qp}(\delta))$ can be designed, then in the algorithm, if q is correct, it must eventually send messages at every clock tick.*

4.2 Approximation algorithms

Algorithm assuming perfectly synchronized clocks. We first assume that processes are equipped with perfectly synchronized clocks denoted $clock()$, *i.e.*, for every time τ, every process p and q, we have $clock_p() = clock_q()$ at time τ. Considering the link (q, p), a constant $K > 1$, algorithm given in Figure 4 allows process p to *approximate* $(T_{qp}(\delta), T_{qp}(\delta + K))$. In the algorithm, process p has a boolean variable $Out_{\mathcal{A}_{(q,p)}}$ that is initialized to $true$ and reset to $true$ every η time. This variable remains $true$ until p learns that a message from q to p may take more than $\delta + K$ time units. In this case $Out_{\mathcal{A}_{(q,p)}}$ is set to $false$. If in the extraction algorithm, p waits that $Out_{\mathcal{A}_{(q,p)}}$ is false, then it is notified.

To test the timeliness of the link, we proceed as follows: every K time, q sends to p a `TIMELY?` message where it stores the current value of its local clock. As the clocks are perfectly synchronized, upon receiving a `TIMELY?` message tagged with the clock value c, p knows the exact time the message spends to traverse the link. If the message spends more that δ time units, p has an evidence that was not timely and, consequently, sets $Out_{\mathcal{A}_{(q,p)}}$ to $false$. Moreover, we use a *timer* of period $K + \delta$. If p does not receive any message from q during a period of $K + \delta$ time units, p suspects q and consequently sets $Out_{\mathcal{A}_{(q,p)}}$ to $false$.

If the link (q, p) is δ-timely, then q is correct. Hence, q sends `TIMELY?` messages to p infinitely often and eventually all these messages are received by p on time. So, eventually p stops setting $Out_{\mathcal{A}_{(q,p)}}$ to $false$. Hence, $Out_{\mathcal{A}_{(q,p)}}$ is eventually forever true.

If the link (q, p) is not $(\delta + K)$-timely, there is two cases to consider:

- If q eventually crashes, the timer guarantees that $Out_{\mathcal{A}_{(q,p)}}$ is $false$ infinitely often.
- Assume now that q is correct. If the link is not $(\delta + K)$-timely then for infinitely many time τ, $CT(\tau, q, p) > \delta + K$. Let τ be such a time. There exists τ' such that process q sends (`TIMELY?`) messages at τ' and at $\tau' + K$ with $\tau' < \tau \leq \tau' + K$. By the FIFO property on the links, the message sent at $\tau' + K$ is received after $\tau + \delta + K + 1 > (\tau' + K) + \delta + 1$. Hence, $Out_{\mathcal{A}_{(q,p)}}$ is set to $false$.

In both cases $Out_{\mathcal{A}_{(q,p)}}$ is $false$ infinitely often.

Lemma 1. *If processes are equipped with perfectly synchronized clocks, algorithm given in Figure 4 is an* approximation *algorithm of $(T_{qp}(\delta), T_{qp}(\delta + K))$ for process p.*

With Proposition 1, we obtain:

Theorem 1. *If processes are equipped with perfectly synchronized clocks, there is an algorithm for all processes to approximate $(T_{qp}(\delta), T_{qp}(\delta + K))$.*

```
Code for process $p$
Initialization
    $Out_{\mathcal{A}_{(q,p)}} \leftarrow true$
    settimer($qp$) $\leftarrow K + \delta$
    start Task 1, Task 2 and Task 3

Task 1
    upon receive⟨TIMELY?, $c$⟩ from $q$ do
        settimer($qp$) $\leftarrow K + \delta$
        if clock() $- c > \delta$ then
            $Out_{\mathcal{A}_{(q,p)}} \leftarrow false$     /* the extraction algorithm at $p$ will notice that $Out_{\mathcal{A}_{(q,p)}} is false$ */

Task 2
    upon timerexpire($qp$) do
        settimer($qp$) $\leftarrow K + \delta$
        $Out_{\mathcal{A}_{(q,p)}} \leftarrow false$     /* the extraction algorithm at $p$ will notice that $Out_{\mathcal{A}_{(q,p)}} is false$ */
Task 3
    do every $\eta$ time     /* $\eta$ is a constant */
        $Out_{\mathcal{A}_{(q,p)}} \leftarrow true$

Code for process $q$
Initialization
    start Task 4

Task 4
    do every $K$ time
        send⟨TIMELY?, clock()⟩ to $p$
```

Fig. 4. Algorithm for p to approximate $(T_{qp}(\delta), T_{qp}(\delta + K))$, assuming perfectly synchronized clocks.

Algorithm assuming a Γ-timely path in the reverse side. We now assume that local clocks may not be synchronized. Instead, we assume that if the link (q, p) is δ-timely, then there exists a Γ-timely path from p to q, that is, a path such that if p and q are correct and p sends a message to q at time τ, then there exists some correct processes r_1,...,r_k and some time τ_1,...,τ_k such that (1) $r_1 = p$, (2) $r_k = q$, (3) $\tau_1 = \tau$, (4) $\tau_k \leq \tau + \Gamma$, and (5) for all $1 \leq i < k$, if r_i sends a message at time τ_i, it is received by r_{i+1} at time τ_{i+1}.

Considering the link from q to p, the algorithm given in Figure 5 allows process p to *approximate* the predicate $(T_{qp}(\delta), T_{qp}(2\delta + \Gamma + K))$.

In the algorithm, process p has a boolean variable $Out_{\mathcal{A}_{(q,p)}}$ that behaves as in the previous algorithm.

To test the timeliness of (q, p), we proceed by phases. Every K times, p broadcasts to every other process a TIMELY? message where it stores the current phase number and a counter value initialized to 0. Then, using the counter, the message is relayed at most $n - 1$ times in all directions except p until reaching q. These relays guarantee that if there exists a Γ-timely path from p to q, then at least one TIMELY? message tagged with the current phase number arrives to q in less than Γ time units. Upon receiving such a message, q relays the message a last time to p only. Hence, if (q, p) is δ-timely, p receives from q at least one TIMELY? message tagged with the current phase number in less than $\delta + \Gamma$ time units. If p does not receive this message by time $\delta + \Gamma$, it sets $Out_{\mathcal{A}_{(q,p)}}$ to $false$. To this end, it uses one timer per phase that is activated when it sends a TIMELY? message and this timer is disabled if the corresponding TIMELY? message is received from q on time.

```
Code for process $p$
1: **Initialization**
2:     $phase\_qp \leftarrow 0$
3:     $Out_{\mathcal{A}_{(q,p)}} \leftarrow true$
4:     start Task 1, Task 2, Task 3 and Task4

5: **Task 1**
6:     **do every** $K$ **time**
7:         $phase\_qp \leftarrow phase\_qp + 1$
8:         send$\langle$TIMELY?, $phase\_qp$, 0$\rangle$ **to** every process except $p$
9:         settimer($phase\_pq$) $\leftarrow \delta + \Gamma$

10: **Task 2**
11:     **upon** receive$\langle$TIMELY?, $\ell$, $-\rangle$ **from** $q$
12:         unsettimer($\ell$)

13: **Task 3**
14:     **upon** timerexpire($\ell$) **do**
15:         $Out_{\mathcal{A}_{(q,p)}} \leftarrow false$     /* the extraction algorithm at $p$ will notice that $Out_{\mathcal{A}_{(q,p)}} is false$ */

16: **Task 4**
17:     **do every** $\eta$ **time**     /* $\eta$ is a constant */
18:         $Out_{\mathcal{A}_{(q,p)}} \leftarrow true$

Code for process $q$
1: **Initialization**
2:     start Task 5

3: **Task 5**
4:     **upon** receive$\langle$TIMELY?, $t$, $k\rangle$ **from** any process $r$ **do**
5:         send$\langle$TIMELY?, $t$, $k+1\rangle$ **to** $p$

Code for every process except $p$ and $q$
1: **Initialization**
2:     start Task 6

3: **Task 6**
4:     **upon** receive$\langle$TIMELY?, $t$, $k\rangle$ **from** any process $r$ **do**
5:         **if** $k \leq n-1$ **then**
6:             send$\langle$TIMELY?, $t$, $k+1\rangle$ **to** every process except $p$
```

Fig. 5. Algorithm for p to *approximate* $(T_{qp}(\delta), T_{qp}(2\delta+\Gamma+K))$ assuming a Γ-timely path in the reverse side.

If the link (q,p) is δ-timely, then q is correct. So, there is a time after which, during every phase, p receives at least one TIMELY? message from q in less than $\delta+\Gamma$ time. Hence, $Out_{\mathcal{A}_{(q,p)}}$ is eventually forever true.

If the link (q,p) is not $(2\delta+\Gamma+K)$-timely, there is two cases to consider:

- If q eventually crashes, eventually p stops receiving TIMELY? messages and, consequently, sets $Out_{\mathcal{A}_{(q,p)}}$ to $false$ infinitely often.
- Assume now that q is correct. If the link is not $(2\delta+\Gamma+K)$-timely then for infinitely many time τ, $CT(\tau,q,p) > 2\delta+\Gamma+K$. Let τ be such a time.
  By definition, if q sends a message to p at time τ, then p receives the message at a time τ' such that $\tau' > \tau+2\delta+\Gamma+K$. Moreover, the link being FIFO, (1) every message sent by q to p after time τ is received after time $\tau' > \tau+2\delta+\Gamma+K$.
  Every K time, p increments ph, sends a message (TIMELY?,ph,0), and starts a timer identified by ph that will expire $\delta+\Gamma$ times later. In particular, p sends such a message, say (TIMELY?,ph_i,0), at a time τ_i such that $\tau \leq \tau_i \leq \tau+K$ and starts a timer ph_i that will expire at the latest at time $\tau+\delta+\Gamma+K$. Moreover, (2) before time τ_i no (TIMELY?,ph_i,$-$) message exists in the system. So by (1) and (2), p cannot receive any (TIMELY?,ph_i,$-$) message before time $\tau' > \tau+2\delta+\Gamma+K$. Hence, p cannot receive any (TIMELY?,ph_i,$-$) message before timer ph_i expires. Consequently, p will set $Out_{\mathcal{A}_{(q,p)}}$ to $false$. Hence, if (q,p) is not $(2\delta+\Gamma+K)$-timely, p sets $Out_{\mathcal{A}_{(q,p)}}$ to $false$ infinitely often.

In both cases $Out_{\mathcal{A}_{(q,p)}}$ is $false$ infinitely often.

Lemma 2. *If there is a Γ-timely path from p to q, algorithm given in Figure 4 is an* approximation *algorithm $(T_{qp}(\delta), T_{qp}(2\delta + \Gamma + K))$ for process p.*

With Proposition 1, we obtain:

Theorem 2. *If there is a Γ-timely path from p to q, then there is an algorithm for all processes to approximate $(T_{qp}(\delta), T_{qp}(2\delta + \Gamma + K))$*

δ-timely paths In the same way it possible to approximate a δ-timely path P from p to q, (every message sent along the path arrives to process q within δ), we denote this predicate $T_P(\delta)$. The algorithms that approximate $(T_P(\delta), T_P(\Delta))$ are similar to those Figure 4 and Figure 5. Let P be a path from p to q, we have:

Theorem 3. *If processes are equipped with perfectly synchronized clocks, there is an algorithm for all processes to approximate $(T_P(\delta), T_P(\delta + K))$.*

Theorem 4. *If there is a Γ-timely path from q to p, then there is an algorithm for all processes to approximate $(T_P(\delta), T_P(2\delta + \Gamma + K))$*

5 Extraction of δ-timeliness graphs

In this section, we apply our previous results to extract graphs. To that end, we give few examples of sets of predicates on graphs. These predicates concern the δ-timeliness of the edges of the graph. We show that we can extract an element of such sets under some assumptions. We also exhibit some desirable properties of the extracted graphs.

We begin with some definitions and notations about graphs.

5.1 Graphs

For a directed graph $G = \langle N, E \rangle$, $Nodes(G)$ and $Edges(G)$ denote N and E, respectively. The tuple (X, Y) is a *directed cut* (*dicut* for short) of G if and only if X and Y define a partition of $Nodes(G)$ and there is no directed edge $(y, x) \in Edges(G)$ such that $x \in X$ and $y \in Y$.

Lemma 3. *For every path $p_0, \ldots p_m$ constituted of Δ-timely links, there exists $k \in [0 \ldots m+1]$ such that:*

- *for all j such that $0 \leq j < k$, p_j is a correct process, and*
- *for all j such that $k \leq j \leq m$, p_j is faulty.*

Proof: For all $i \in [0 \ldots m-1]$, (p_i, p_{i+1}) is Δ-timely and, by definition, if p_{i+1} is correct then p_i is also correct, which proves the lemma. $\Box$

We deduce the following corollary from Lemma 3:

Corollary 1. *Let P be a path from p to q constituted of Δ-timely links.*

- *If p and q are correct, then all processes in P are correct.*
- *If P is a cycle, then either all processes are correct or all processes are faulty.*

5.2 Extracting an elementary δ-timely path from p to q

Let $(Path_i)_{i \in I}$ be the set of all possible elementary paths from p to q. If $(\mathcal{A}_i)_{i \in I}$ is a set of approximation algorithms of $(T_{Path_i}(\delta)), T_{Path_i}(\Delta))_{i \in I}$, then from Proposition 3, there is an extraction algorithm for $(T_{Path_i}(\delta), T_{Path_i}(\Delta))_{i \in I}$, and from Proposition 4, there is a communication-efficient extraction algorithm for $(T_{Path_i}(\delta), T_{Path_i}(\Delta))_{i \in I}$. By Proposition 2 and Theorem 3, assuming perfectly synchronized clocks, there is an approximation algorithm for $(T_{Path_i}(\delta), T_{Path_i}(\Delta))_{i \in I}$ for every $\Delta > \delta$. Hence, we can conclude:

Theorem 5. *Assuming perfectly synchronized clocks and $\Delta > \delta$, there exists a (communication-efficient) extraction algorithm for $(T_{Path_i}(\delta), T_{Path_i}(\Delta))_{i \in I}$.*

Following a similar reasoning, using Theorem 2, we have:

Theorem 6. *Assuming a Γ-timely path in the reverse side, if $\Delta > 2\delta + \Gamma$, there is a (communication-efficient) algorithm for $(T_{Path_i}(\delta), T_{Path_i}(\Delta))_{i \in I}$.*

By Corollary 1, if p and q are correct, then every Δ-timely path from p to q only contains correct processes. Hence, the algorithm we obtained allows to efficiently route message in the network. Moreover, our approach being modular, one can design a routing algorithm for only a restricted subset of processes. For example, if we consider a clustered network, one may want to design an efficient routing algorithm only for the set of clusterheads (the communication inside a cluster being usually local or managed using an overlay like a tree).

5.3 Extracting δ-timely graphs

We want now to extract δ-timely graphs containing all correct processes. Below, we give some definitions to formally explain our approach.

Consider the set of all graphs $(G_i)_{i \in I}$ that can be constructed with $Nodes(G_i) \subseteq \Pi$ and $Edges(G_i) \subseteq Nodes(G_i) \times Nodes(G_i)$.

$TG_{G_i}(\delta)$ is true in run R if and only if eventually (1) all nodes of $\Pi - G_i$ are crashed, and (2) all edges (p, q) of G_i are δ-timely.

By definition, if $TG_{G_i}(\delta)$ is true in a run R, G_i contains all correct processes. If $(\mathcal{A}_i)_{i \in I}$ is a set of approximations of $(TG_{G_i}(\delta), TG_{G_i}(\Delta))_{i \in I}$, then by Propositions 2, 3, and 4, there is an (efficient) extraction algorithm for $(TG_{G_i}(\delta), TG_{G_i}(\Delta))_{i \in I}$.

To define $TG_{G_i}(\delta)$ we need a local predicate $P_{p \ is \ crashed}$ that states if a process p is crashed. A local algorithm that approximates this predicate is given below.

Approximate crashed processes. We can easily design an approximation algorithm $\mathcal{A}_p$ for $(P_{p \ is \ crashed}, P_{p \ is \ crashed})$ at every process $q \neq p$: process p regularly sends messages. Each time a process receives such a message it sets $Out_{\mathcal{A}_p}$ to false. Moreover, q regularly resets $Out_{\mathcal{A}_p}$ to $true$. If p is faulty, $Out_{\mathcal{A}_p}$ is eventually forever $true$. Otherwise (p is correct), $Out_{\mathcal{A}_p}$ is infinitely often $false$. Hence, follows:

Proposition 7. *Algorithm given in Figure 6 allows every process q to* approximate $(P_{p \ is \ crashed}, P_{p \ is \ crashed})$.

Using $P_{p \ is \ crashed}$ and $T_e(\delta)$, we can now define $TG_{G_i}(\delta)$ as follows: $TG_{G_i}(\delta) \equiv \bigwedge_{e \in Edges(G_i)} T_e(\delta) \wedge \bigwedge_{v \notin Nodes(G_i)} Crash(v)$.

Assuming perfectly synchronized clocks, by Propositions 7, 2, and Theorem 1, there is an approximation algorithm $\mathcal{A}_{G_i}$ for $(TG_{G_i}(\delta), TG_{G_i}(\Delta))_{i \in I}$ if $\Delta > \delta$. By Propositions 3 and 4, we can conclude:

```
Code for process $p$
Initialization
    start Task 1

Task 1
    do every $\eta$ time        /* $\eta$ is a constant */
        send$\langle (ALIVE) \rangle$ to every process except $p$
        $Out_{\mathcal{A}_p} \leftarrow false$        /* the extraction algorithm at $p$ will notice that $Out_{\mathcal{A}_p}$ is $false$ */

Code for process $q \neq p$
Initialization
    start Task 2 and Task 3

Task 2
    upon receive$\langle$ALIVE$\rangle$ from $p$ do
        $Out_{\mathcal{A}_p} \leftarrow false$
Task 3
    do every $\eta$ time        /* $\eta$ is a constant */
        $Out_{\mathcal{A}_p} \leftarrow true$
```

Fig. 6. Algorithm to approximate $(P_{p\ is\ crashed}, P_{p\ is\ crashed})$.

Theorem 7. *Assuming perfectly synchronized clocks and $\Delta > \delta$, there exists a (communicationn-efficient) algorithm that extracts $(TG_{G_i}(\delta), TG_{G_i}(\Delta))_{i \in I}$.*

Following a similar reasoning, by Proposition 2, 3, 4, 7, and Theorem 2, we have:

Theorem 8. *Assuming a Γ-timely path in the reverse side, if $\Delta > 2\delta + \Gamma$, there is a (communication-efficient) algorithm that extracts $TG_{G_i}(\delta), TG_{G_i}(\Delta))_{i \in I}$.*

From the previous theorem and Corollary 1, we can deduce the next corollary, which states that the extracted graph is a dicut between correct and faulty processes. Note that this property is very useful. For example, one can design an approximation algorithm to extract a tree. Then, if not all the processes are faulty, the extracted tree will be rooted at a correct process, the tree will contain all correct processes, and in the tree there will exist a Δ-timely path from the root to every other correct process. Hence, the algorithm will allow to communication-efficiently route message from a correct to all others.

Corollary 2. *If G_{i_0} is the extracted graph, $G_{i_0}[Correct(R)]$, $G_{i_0}[Faulty(R)]$ is a directed cut of G_{i_0}*

From Corollaries 1 and 2, we have the next corollary, which gives a sufficient condition to evaluate $\Diamond\mathcal{P}$, the eventually perfect failure detector [5] that eventually outputs exactly the set of correct processes.

Corollary 3. *If there is at least one correct process and if $(G_i)_{i \in I}$ contains only strongly connected graphs, the extracted graph G_{i_0} contains only correct processes.*

Extracting a ring containing all correct processes. We now want to extract a δ-timely ring among all correct processes. Consider the set of all graphs $(Ring_i)_{i \in I}$, that are all possible rings among any non-empty subset of Π.

$TG_{Ring_i}(\delta)$ is true in run R if and only if eventually (1) all nodes of $\Pi - G$ are crashed, and (2) all edges (p, q) of G are δ-timely.

By Corollary 1, if $Ring_G(\delta)$ is true in run R then G contains exactly the set of correct processes of run R.

Assuming perfectly synchronized clocks, by Propositions 2, 7, and Theorem 1, there is an approximation algorithms for $(TG_{Ring_i}(\delta), TG_{Ring_i}(\Delta))_{i \in I}$. By Propositions 3 and 4, we can conclude:

Theorem 9. *Assuming perfectly synchronized clocks, if $\Delta > \delta$, there exists a (communication-efficient) extraction algorithm for $(TG_{Ring_i}(\delta), TG_{Ring_i}(\Delta))_{i \in I}$.*

In a ring composed of δ-timely links, each link (p, q) of the ring is δ-timely and there is a path from q to p that is $(n-1)\delta$-timely. Then, by Propositions 2, 3, 4, and Theorem 2 , we have:

Theorem 10. *If $\Delta > (n+1)\delta$, there is a (communication-efficient) extraction algorithm for $(TG_{Ring_i}(\delta), TG_{Ring_i}(\Delta))_{i \in I}$.*

As noted previously the extracted ring contains exactly the correct processes.

6 Concluding remarks

In this paper, we studied in which way processes may approximate and agree on structural properties based on δ-timeliness (*e.g.*, find δ-timely paths between processes or build a ring between correct processes using only δ-timely links).

We focused on δ-timeliness of the links, however other properties are also interesting to approximate. For example, with general timeliness defined as the existence of some unknown bound on communication delays, we can get the same results as in [7]. Approximation and extraction algorithms may be considered as a first step to dynamically evaluate predicates expressed in some kind of temporal logic.

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., Deporte-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. Chu, F.: Reducing Ω to $\Diamond W$. Information Processing Letters 67(6), 298–293 (Sep 1998)
7. Delporte-Gallet, C., Devismes, S., Fauconnier, H., Larrea, M.: Algorithms for extracting timeliness graphs. In: SIROCC0 (2010)
8. 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)
9. 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)
10. 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)
11. Mostéfaoui, A., Mourgaya, E., Raynal, M.: Asynchronous implementation of failure detectors. In: DSN. pp. 351–360. IEEE Computer Society (2003)