WHO'S ON BOARD?
Probabilistic Membership for Real-Time Distributed Control Systems

(Extended)

Rachid Guerraoui
EPFL
rachid.guerraoui@epfl.ch

David Kozhaya
EPFL
david.kozhaya@epfl.ch

Manuel Oriol
ABB Corporate Research
manuel.oriol@ch.abb.com

Yvonne-Anne Pignolet
ABB Corporate Research
yvonne-anne.pignolet@ch.abb.com

***Abstract*—To increase their dependability, distributed control systems (DCSs) need to agree in real time about which hosts have crashed, i.e., they need a real-time membership service. In this paper, we prove that such a service cannot be implemented deterministically if, besides host crashes, communication can also fail. We define implementable probabilistic variants of membership properties, which constitute what we call a *synchronous membership service* (SYMS). We present an algorithm, *ViewSnoop*, that implements SYMS with high-probability.**

We implement, deploy and evaluate *ViewSnoop* analytically as well as experimentally, within an industrial DCS framework. We show that *ViewSnoop* significantly improves the dependability of DCSs compared to membership schemes based on classic heartbeats, at low additional cost. Moreover, *ViewSnoop* distinguishes, with high probability, host crashes from message losses, enabling DCSs to counteract losses better than existing approaches.

I. INTRODUCTION

Among the many financial hazards of industrial plants, *downtime* (the time during which a plant's normal operation is halted) is one of the most expensive ($\approx$ 12,500\$$/hr$) [1]. *Automated control systems*, which manage these plants, hence require increased dependability.

An automated control system comprises a set of control applications; each being a program that handles (parts of) an industrial system and generally adheres to hard real-time constraints. In order to tolerate crashes, control systems are often decentralized [2]–[4]. Such *distributed control systems* (DCSs) require however to learn about hosts (processes) which have crashed in order to initiate proper recovery measures.

A. Distributed control systems (DCSs)

Most control applications running on DCSs are cyclic [5]–[7]. Such applications consist of several small *tasks* that execute periodically. Some of these tasks run concurrently on several hosts, possibly on behalf of different control applications. A *scheduler*, a distributed DCS module, typically maps tasks to non-crashed hosts and specifies the order in which these tasks execute (Figure 1(a)).

A DCS cannot avoid host crashes and message losses. Control systems typically experience host crash rates of about 10^{-5}/hr and message loss rates in the range of $10^{-5}/hr$ (permanent losses) and $10^{-3}/hr$ (transient losses) [8], [9]. In order to "recover" from crashed hosts, DCSs need to know about these crashes and react in real-time; for example, to have the scheduler re-map tasks to non-crashed hosts, ensuring proper execution of all applications [10], [11] (see Figure 1(b)). Besides the real-time necessity, hosts in DCSs need to have consistent views of which hosts in the system have crashed. Inconsistent views imply that hosts might consider different

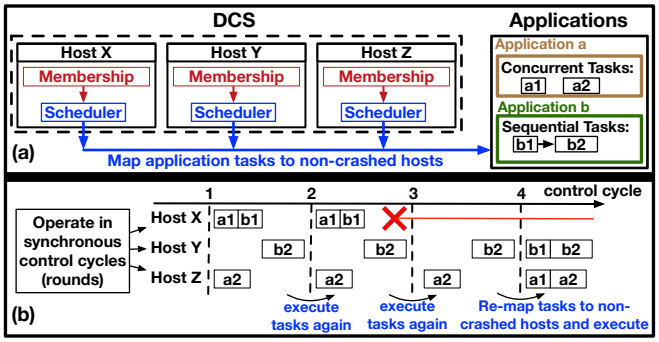

Figure 1. A DCS with three hosts running two control applications.

hosts as crashed. The state of the scheduler module in a DCS, as a result, might be invalidated yielding improper executions of applications and causing downtime [7]. Basically, a DCS needs a *group membership service* [12]–[18] to coordinate the information regarding crashed hosts in real time.

B. Membership in a DCS

In short, an ideal membership in DCSs amounts to a service that synchronously reports, to all (non-faulty) hosts, perfect information about host crashes and within fixed bounds (see Figure 2). In the presence of message losses, however, implementing such a service deterministically is impossible as we show in this paper (Section III).

In fact, known implementations of deterministic membership services either assume no message losses, provide eventual (not real-time) guarantees or use additional help [18]–[22]. The implementable membership guarantees in contexts similar to DCSs can, at best, be probabilistic [8], [13], [23]. We define accordingly a *synchronous membership service* (SYMS), a new abstraction encapsulating a probabilistic form of the ideal membership properties needed by DCSs.

We propose *ViewSnoop*, a new algorithm that ensures SYMS properties with high probability (relative to other membership mechanisms and which persists with increasing system size, see Section V). The main idea underlying *ViewSnoop* is to (a) have hosts maintain local suspicion lists that are not visible to the scheduler of a DCS and at the same time (b) let hosts snoop into each others' local views by modifying the structure of heartbeats, precisely by piggybacking local suspicion lists on heartbeats. Heartbeats are disseminated periodically via a broadcast primitive (not necessarily reliably) as in most DCSs to facilitate crash detection [7], [8], [12]–[14]. Appending suspicion lists to heartbeats helps hosts know about other alive hosts, despite possible message losses. This property increases the probability of having a global consistent view and hence a better accuracy. Combined with (a), *ViewSnoop* can, with high

probability, discern message losses from host crashes, better than using sequence numbers [24] (see Section V). Having losses mistaken for host crashes, and removing correct hosts as a result, not only worsens accuracy[1], but also depletes processing resources, threatening availability.

C. Results

We first evaluate analytically the performance metrics of *ViewSnoop*. We compare with membership services based on classic heartbeats [7], [8], [12], [13]. We show that:

1. *ViewSnoop* provides better guarantees, on both view agreement among hosts and accuracy, compared to membership services based on classic heartbeats alone, e.g., 9.2x better agreement probability and 1.6x better accuracy probability for a system with 10 hosts. This improvement increases exponentially and un-boundedly with system size.

2. *ViewSnoop* distinguishes host crashes from message losses, without jeopardizing accuracy (which all membership services based on classic heartbeats suffer from). *ViewSnoop*, thus, allows better configurations to be computed, accounting for bad links rather than excluding correct hosts.

We then report on a full implementation of *ViewSnoop* in an industrial DCS framework, called FASA [7]. We evaluate *ViewSnoop*'s performance experimentally (on FASA), comparing it to a classic heartbeat-based implementation, deployed in most existing DCS frameworks [7], [8], [12], [13].

We show experimentally that *ViewSnoop* is significantly more dependable than the classic heartbeat-based implementation. More precisely, *ViewSnoop* provides a higher accuracy, ranging from 2.5x up to 4x better than the classic implementation. The higher the accuracy, the fewer correct hosts are excluded. Thus, the risk of downtime, due to the lack of processing resources, becomes smaller. This improvement increases as the system size grows.

We also assess the trade-offs underlying *ViewSnoop*'s design and implementation:

1. *ViewSnoop* notifies hosts about crashes in real-time, with a lower downtime risk compared to using classic heartbeats. The trade-off is only one extra control cycle (or round, see definition in Section II) to recognize crashes and recoveries. *ViewSnoop*, always excludes crashed hosts from the system in less than three cycles after crashing (real-time). *ViewSnoop* also allows recovering hosts, given no message losses, to join the system in less than two cycles. The classic heartbeat-based implementation requires one cycle less. It is crucial to note though, that if this trade-off is eliminated, precisely by allowing classic heartbeat-based memberships for an additional cycle to recognize crashes and recoveries, we show that *ViewSnoop* still provides better accuracy and availability.

2. *ViewSnoop* induces 0.3 μs (7%) processing overhead and 200 bytes/sec (11%) network overhead (for UDP over Ethernet), over the classic heartbeat-based implementation. However, the added delay neither affects FASA, nor the upper layer applications, as *ViewSnoop* fully executes within the idle time of a host, while the network overhead is 1.6% (IPv4) and 1.2% (IPv6) of the packet size being sent in the classic mechanism.

[1]The probability of not excluding non-crashed hosts.

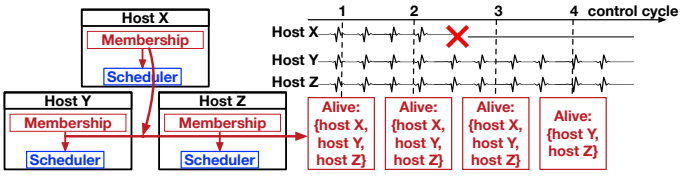

Figure 2. Overview of an ideal membership service for a DCS.

3. *ViewSnoop* periodically broadcasts suspicion lists, rather than broadcasting classic heartbeats. Broadcasting suspicion lists will likely lead to a bandwidth bottleneck in large-scale distributed systems if host crashes and losses are common. Yet, *ViewSnoop* is targeted for DCSs, i.e., environments where crashes and losses typically seldom occur [8], [9].

D. Summary of Contributions

The main contributions of this paper are the following:

1. A specification of the membership requirements for DCSs running cyclic applications. We prove that a deterministic form of these ideal requirements is impossible to implement in a system with both host crashes and message losses. We define SYMS, a probabilistic abstraction of the requirements of DCSs. SYMS can be implemented despite message loss.
2. *ViewSnoop*, an algorithm implementing SYMS with high probability. *ViewSnoop*'s design allows it to distinguish host crashes from message losses, better than using message sequence numbers [24], and without affecting accuracy.
3. An experimental and an analytic evaluation of *ViewSnoop*'s performance showing that *ViewSnoop* provides a significantly more dependable service, enhancing a DCS's availability, compared to methods relying on classic heartbeats [7], [8], [12], [13].

Road-map. Section II recalls the notion of DCSs in more details. Section III identifies the membership requirements for DCSs and proves the impossibility of deterministically implementing an ideal service before introducing SYMS (its probabilistic variant). Section IV presents *ViewSnoop*, our algorithm for implementing SYMS, and computes its probabilistic guarantees. Section V presents an analytic evaluation of the probabilistic guarantees of *ViewSnoop* compared to memberships using classic heartbeats. Section VI discusses platform, application and implementation details. Section VII evaluates our implementation of *ViewSnoop* in FASA, an industrial DCS framework. Section VIII discusses related work and Section IX concludes the paper. For better illustration, we defer some computations to a dedicated Appendix.

II. DCSs For Cyclic Control Applications

A distributed control system (DCS) for cyclic control applications consists of a set of hosts, $\Pi = \{h_1, h_2, ..., h_n\}$, physically mapped to cores of the same or different machines. Clearly, these hosts can *fail (crash)* [8], i.e., stop executing operations. Hosts have access to local synchronized clocks with bounded skew. Accordingly, all hosts define control cycles (rounds) of the same fixed duration. Control cycles are synchronized among hosts, i.e., the start and end of a cycle occur at all hosts at the same time (with a bounded skew). During every control cycle, each host executes the tasks assigned to it by the *scheduler* (recall Figure 1).

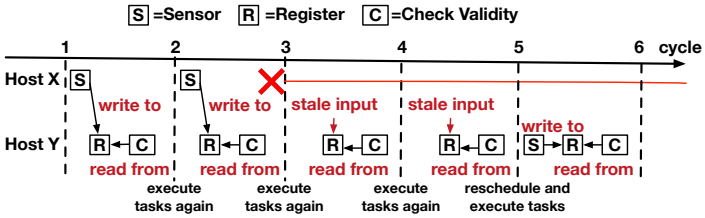

Figure 3. An example of a DCS tolerating two cycles of stale input, which result from the failure of the host responsible to refresh the data.

Scheduler. This is a distributed module that specifies which application tasks run on which hosts and in what order. The allocation of tasks to hosts is called a *configuration*. A scheduler makes sure that all hosts can execute the assigned tasks without exceeding the total cycle duration. Moreover, the scheduler ensures that configurations allow all applications to meet their deadlines (timing constraints). As such, a host requiring some input, in some configuration, expects the value of this input to be refreshed, every cycle (e.g., by another host driving this input, see Figure 3). A value that is not refreshed in time is called a *stale input*. Typically in DCSs, hosts can tolerate to read stale input up to a bounded number of consecutive cycles, say s_c. If the input remains stale for more than s_c cycles, then a new configuration has to be installed.

A DCS requires to exclude a crashed host: (i) within a bounded number of cycles after crashing (real-time) and (ii) synchronously at all alive hosts, i.e., in the same cycle. Violating (ii) might lead the scheduler state to be inconsistent, resulting in hosts executing different configurations (mapping of tasks to hosts). Applications, as a result, might execute incorrectly, as communication and/or the order of execution between tasks of the same application might be invalid.

Communication. Every pair of hosts is connected by two logical uni-directional links. Links here for example abstract a physical bus or a dedicated network link. Arguably, all communication is prone to random disturbances resulting from bad channel quality, collisions, stack overflows etc [25]. Messages can thus be lost. When there is no loss, we assume that messages have a bounded delay, say d. Configurations computed by the scheduler account for the delay d. As such, any message scheduled to be sent in cycle r, if not lost, is assumed to be received in the same cycle r.

Specifically, we model losses as follows: a message sent by h_i is received by h_j with probability p. Sending a message reliably at any point in time from one host to another, thus, can take an unbounded amount of time, due to losses and follow-up re-transmissions. We assume, for the theoretical analysis, that p is independent of time and links and is the same for all links. Nevertheless, correlated losses, although not considered theoretically for the tractability of our analysis, can occur in our experimental evaluation (Section VII).

Crash monitoring. In this paper, we consider monitoring schemes that rely only on message exchange and synchronized local clocks (time-outs). Specifically, a host sends messages, known as heartbeats, every cycle to at least one other host in the system (hosts do not send heartbeats to themselves). We assume no causality between heartbeats sent in the same cycle; the content of heartbeats sent by a host during cycle r is the same and can be affected only by the heartbeats sent at cycles $< r$. A host is "alive" at cycle r, if that host does not crash during r (during r a host is either crashed or alive). Hosts do not crash themselves on purpose, as resources for running tasks become scarcer, risking some applications to halt.

III. Overview of SYMS

After our description of the operation of a DCS, we identify now the ideal membership, following the traditional way of defining its properties [18]–[20]. For simplicity, we specify the properties for host crashes in the fail-stop model [26], i.e., without recovery (we discuss recoveries in Section VII-D).

We first introduce some terminology. We denote a view by the tuple (id, M), where M is the set of hosts declared in a view as alive (not crashed). Variable id denotes the view identifier (namely the cycle in which the view is installed). Initially all hosts install the view $V = (id, M)$, where M includes all hosts in the system. Consequent views are obtained from monitoring, as described in Section II. We assume that if a host h_i receives a heartbeat sent by a host h_j in round r, then h_i cannot exclude[2] h_j in round $r+1$. The ideal properties for DCSs can be expressed as follows:

P1: *Monotonicity:* If a host installs a view $V = (id, M)$ and later $V' = (id', M')$, then $id < id'$ and $M' \subseteq M$.
P2: *Agreement:* If a host installs a view $V = (id, M)$ at round r, then all alive hosts at r install $V = (id, M)$ at r.
P3: *Completeness:* If a host h crashes, then after a maximum of s_c rounds elapse after the crash, all hosts that remain alive s_c rounds after the crash, install $V = (id, M)$ where $h \notin M$.
P4: *Accuracy* If a host installs view $V = (id, M)$ where $h \notin M$, for some host h, then h has already crashed.
P5: *Non-triviality:* Let $\mathcal{C}$ be the set of hosts alive at round r, during which host h installs a view $V = (id = r, M)$. Then, it is possible that $\{\mathcal{C} \cap M'\} \subseteq M$, where $V' = (id' < r, M')$ is the most recent view h installed before the view at r.

In contrast with traditional membership properties [18]–[20], which detect crashes eventually and not necessarily in a synchronous manner (i.e., in the same synchronous round by all hosts), completeness (P3) and accuracy (P2) here stipulate real-time and synchronous detection of crashes respectively.

Theorem 1. *No algorithm can deterministically guarantee both completeness (P3) and accuracy (P4) in a DCS with message losses.*

Proof: Both properties, P3 and P4, are related to failure detection, precisely perfect failure detection[3]. A perfect failure detector [26] cannot be implemented in case of message loss, because finite executions where a host crashes cannot be distinguished from finite executions where all messages from this host are lost [52]. For better illustration, we showcase this fact again below.

Consider an example of a DCS with two hosts, h_1 and h_2, and the following executions:

- $e1$. an execution where host h_2 fails at cycle r.

[2]This assumption is analogous to our assumption that hosts cannot be crashed on purpose, in the sense that, as long as h_i receives heartbeats from h_j, then h_j is certainly alive (at least up to the moment of sending the last heard heartbeat). Acting otherwise might risk the system's availability (higher downtime) as fewer hosts become available.

[3]P3 is a stronger version of the strong completeness property (defined in [26]), as it has a bound on the detection time (s_c control cycles versus eventually). P4 is the strong accuracy property. P3 and P4 together define a stronger version of the perfect failure detector [34] (a perfect failure detector with a bound on detection time).

- $e2$. an execution where host h_1 and h_2 are both correct but lose all messages sent (if any) at all cycles in the range $[r, r + s_c]$.

Since the control cycle duration is fixed, a finite number of messages can be sent during a control cycle, say n_i. Execution $e2$ is valid, since $e2$ can occur with the positive probability, $(1-p)^{n_i(s_c+1)}$, p being the probability that a sent message is successfully received. h_1 cannot monitor h_2 (to know if h_2 is alive) except through message exchange and time-outs (see Section II). With respect to h_1 executions $e1$ and $e2$ are indistinguishable during $[r, r + s_c]$, for any finite value of s_c (since h_1 cannot know if h_2 has failed or all messages from h_2 are lost).

By P3, in execution $e1$ h_1 declares h_2 as failed at most by cycle $r + s_c$. Since $e1$ and $e2$ are indistinguishable during $[r, r+s_c]$ then h_1 declares h_2 as failed at most by cycle $r+s_c$ also in $e2$. This violates P4 in execution $e2$. ■

Theorem 2. *No algorithm satisfying non-triviality (P5) can deterministically guarantee agreement (P2) and completeness (P3) in a DCS with message losses.*

Proof: Assume by contradiction that an algorithm $\mathscr{A}$ satisfies the non-triviality property (P5) and deterministically guarantees agreement (P2) and completeness (P3) in a DCS with losses.

Hosts in algorithm $\mathscr{A}$ install an initial view as specified by our assumption in Section III, i.e., a view in which no host is excluded.

Lemma 1. *Assuming that no hosts have been excluded, an algorithm $\mathscr{A}$ satisfying P5 means that exactly one of the following cases is true.*

For every host h_j such that $h_j \in \mathcal{C}$, a host h_i installing a view $V(id, M)$ can:

> ***Case 1.*** *Decide if $h_j \in M$ regardless of any received heartbeats. In this case h_i can decide to*
>
> a) *Include h_j in all views installed.*
> b) *Exclude h_j in a randomly chosen cycle.*
>
> ***Case 2.*** *Decide whether $h_j \in M$ depending on the heartbeats received by h_i.*

Proof: Any view to be installed by any host has to be constructed by the monitoring scheme depicted in Section II.

Assuming that no hosts are excluded (precisely, that no views besides the initial one are installed), P5 can be restated as follows:

Consider some round r in which a host h_i in $\mathscr{A}$ installs a view $V(r, M)$. Then, there is a positive probability that $C \subseteq M$, where $\mathcal{C}$ is the set of hosts alive at round r.

For every host h_j such that $h_j \in \mathcal{C}$, h_i can:

> **Case 1.** Decide if $h_j \in M$ regardless of any received heartbeats. In this case h_i can decide to
>
> a) Include h_j in all views installed.
> b) Exclude h_j in a randomly chosen cycle.
> c) Exclude h_j deterministically in some predetermined cycle r'.
>
> **Case 2.** Decide whether $h_j \in M$ depending on the heartbeats received by h_i.

h_i cannot decide based on any other means, as Section II constrains monitoring to be solely based on exchanging heartbeats.

In fact, having h_i decide according to case 1(c) violates P5. Assume that h_j is a correct host, i.e., a host that does not crash. Then having h_i exclude h_j at any cycle r' deterministically, regardless of any received heartbeats, means that the view at r' can never include h_j as alive, which contradicts P5. As such, in order to satisfy P5, one of the other statements should be true. ■

We will now show that in a system with two hosts h_1 and h_2, deciding according to Case 1 or Case 2 will not allow P2 and P3 to be satisfied deterministically.

If Case 1(a) is true, then $\mathscr{A}$ would violate completeness (P3): Consider an execution where h_1 is correct, i.e., does not crash during the entire execution of the algorithm, and h_2 crashes at some point. If Statement 1(a) is true, then h_1 would never declare h_2 as crashed.

If Case 1(b) is true then $\mathscr{A}$ might violate agreement (P2): Consider an execution where both hosts h_1 and h_2 are alive. Let r' be the cycle in which h_1 excludes h_2 from its view and let r'' be the cycle in which h_2 excludes itself from its own view. Since r' and r'' are randomly chosen, there is a positive probability that $r' \neq r''$. In that case, hosts install different views and P2 is violated.

If Case 2 is true, the following is a necessary condition to satisfy completeness (P3): a host in $\mathscr{A}$ should exclude some host after not hearing (directly or indirectly) from that host for d_t consecutive cycles, such that $d_t \leq s_c$ (given that every host sends a heartbeat at every cycle to at least one other host). According to our monitoring assumptions in a DCS (Section II) any host sends heartbeats to at least one other host in the system.

In a DCS with two hosts, this means that: in every cycle, h_1 sends heartbeats to h_2 and h_2 sends heartbeats to h_1. In other words, h_1 can receive heartbeats only from h_2 and h_2 can receive heartbeats only from h_1.

Consider the case where h_2 loses all heartbeats sent by h_1 for more than s_c consecutive cycles (which can happen with positive probability). By the completeness property (P3), h_2 should install (after d_t consecutive cycles of loss) a view V excluding h_1. Also, h_2 has to decide whether to include or exclude itself from V. The decision taken by h_2, whether to exclude itself from V or not in that case, is independent of what happens to the heartbeats sent by h_2 to h_1 in the past d_t cycles (i.e., if these heartbeats are lost or not). This statement is valid since in that duration h_2 did not receive any heartbeats and thus cannot know any information. Similarly, such a scenario can also happen with h_1. Let us refer to such a scenario, which can occur with either hosts, as Scenario S.

A host in scenario S installs a view V (excluding the other host) and decides either to exclude itself from V or not. We discuss both cases below.

1) A host decides to exclude itself in scenario S. Consider an execution e satisfying both conditions below:

a. h_1 and h_2 correct, i.e., never fail.
b. Starting from cycle r, all heartbeats sent by h_1 to h_2 are lost for $\alpha \cdot s_c$ cycles, i.e., for all cycles in

$[r, r + \alpha \cdot s_c]$, $\forall \alpha \geq 1$, while all heartbeats sent by h_2 to h_1, in this same interval, are not lost.

Condition (b) can happen with positive probability (see proof of Theorem 1). In execution e, h_2 cannot hear any heartbeats in $[r, r + \alpha \cdot s_c]$. In this case and by the completeness property (P3), h_2 excludes h_1 and itself at cycle $r + d_t$.

We recall now the following assumption of Section III: if a host h_i receives a heartbeat sent by host h_j at round r, then h_i cannot exclude h_j in round $r + 1$. Since h_1 receives all heartbeats from h_2 (regardless of the content of these heartbeats) in $[r, r + \alpha \cdot s_c]$, h_1 would still include h_2 as alive during cycle $r + d_t$, which violates agreement (P2).

It is very important to note that in $[r, r + d_t]$, h_2 stops obtaining any additional information. This is due to the fact that h_2 receives no heartbeats in that interval. Thus, at beginning of cycle r, h_2 already has all the information it needs upon which it can base its decision of whether to include or exclude itself from the view. Since execution e does not make any assumptions about heartbeats prior to cycle r, this means that the decision of h_2 to exclude itself in e covers all the possible cases in which h_2 might decide to exclude itself.

2) A host decides not to exclude itself in scenario S. Consider now an execution e' satisfying both conditions below:

a. h_1 and h_2 correct, i.e., never fail.
b. Starting from cycle r, all heartbeats sent by h_1 to h_2 are lost for $\alpha \cdot s_c$ cycles and all heartbeats sent by h_2 to h_1 are lost for $\alpha \cdot s_c$ cycles, i.e., for all cycles in $[r, r + \alpha \cdot s_c]$, $\forall \alpha \geq 1$.

Condition (b) can happen with positive probability (this can be inferred from the proof of Theorem 1 and the fact that losses are independent of links). In execution e', h_2 cannot hear any heartbeats in $[r, r+\alpha \cdot s_c]$. In this case and by the completeness property (P3), h_2 excludes h_1 at cycle $r+d_t$; hence h_2 installs a view at cycle $r + d_t$ where h_2 considers only itself as alive.

Similarly, also by the completeness property (P3) and the fact that a host in scenario S decides not to exclude itself, h_1 installs at cycle $r + d_t$ a view where h_1 is only alive, which violates agreement (P2).

In $[r, r+d_t]$, h_2 stops obtaining any additional information (h_2 receives no heartbeats in that interval). Thus, at beginning of cycle r, h_2 already has all the information it needs upon which it decides to include or exclude itself. The same applies for h_1. Since execution e' does not make any assumptions about heartbeats prior to cycle r, this means that the decision of h_2 not to exclude itself in e' covers all the possible cases in which h_2 might decide not to exclude itself. Combined with the previous case (the case in which h_2 excludes itself at $r+d_t$) we cover all possible cases that might affect h_2's decision.

The same can be constructed for h_1. This result concludes the proof as it shows that an algorithm that satisfies P5 and P3 has a positive probability of violating agreement (P2), given the monitoring families considered in this paper (Section II). ■

Both theorems hold even if only one host can crash.

Given these impossibilities, an implementable form of the desired ideal properties can only be probabilistic. We define such a probabilistic form under an abstraction we call SYMS:

SYMS 1, SYMS 3 and SYMS 5 : respectively as P1, P3 and P5 above.
SYMS 2: If some host installs view $V = (id, M)$ at round r, then with probability p_{agree}, all alive hosts at r install $V = (id, M)$ at round r.
SYMS 4: If some host installs view $V = (id, M)$ such that $h \notin M$, for some host h, then with probability $p_{accurate}$, h has already crashed.

We highlight two probabilistic metrics. The first is p_{agree}, the probability that all alive hosts agree on the view (list of hosts that are considered alive) to be installed at a round. The second is $p_{accurate}$, the probability of an excluded host to have actually crashed. To increase the dependability of a DCS, SYMS algorithms need to maximize both metrics.

IV. The *ViewSnoop* Algorithm

ViewSnoop implements SYMS by building local suspicion lists above which membership views are constructed . Suspicion lists combined with the process of constructing views allow *ViewSnoop* to detect and act upon stale input resulting from message losses and not only host crashes (details in Section V-C). In particular, *ViewSnoop* seeks to increase the probability of having synchronous consensus on views given message losses and to always detect host crashes in real-time.

Let n_i be the maximum number of heartbeats a host h_i can send in some cycle r (every heartbeat, if not lost, is received in r). For illustration, we assume that n_i is the same at all cycles and for all hosts. We first describe *ViewSnoop* for $s_c = 3$ (consecutive cycles in which a host can tolerate stale data); $s_c = 3$ represents the minimum upper bound on the number of cycles to exclude a crashed host in *ViewSnoop* (as we show below). Later, in Section IV-B, we discuss how to extend *ViewSnoop* to any value of $s_c \geq 3$.

Every host in *ViewSnoop* maintains a list of suspected hosts ($local_{suspect}$). In each cycle, every host broadcasts a copy of its suspicion list tagged with the control cycle number, as a heartbeat, n_i times to all hosts. At the end of the cycle the list for the next cycle is prepared. In *ViewSnoop*, a view installed at cycle r has $id = r$.

A. ViewSnoop's Functionalities

1) Synchronous View Agreement. This functionality of *ViewSnoop* constructs the view that a host installs in a control cycle. At the end of the control cycle (i.e., after broadcasting the suspicions list), every host performs a *merge* on all the suspicion lists received: the result is a new view to be installed at the beginning of the next cycle. For every host h_j in the current view, a host h_i performs the merge as follows:

If h_j belongs to the $local_{suspect}$ list of h_i and h_j is in the suspected list of all heartbeats received by h_i, then h_i excludes h_j from the view to be installed in the following cycle. Otherwise h_i considers h_j alive.

Consider a host h_j that belongs to the $local_{suspect}$ list of all alive hosts at cycle r. Then the merge guarantees the following: all alive hosts at cycle $r + 1$ exclude h_j. The reason is that alive hosts at $r + 1$, can receive messages (if any is received) of hosts which append their $local_{suspect}$ lists at r.

2) Host Crashes Detection in Real-time. ViewSnoop aims at excluding crashed hosts in real-time, i.e., within a fixed number of cycles, s_c. This second functionality of *ViewSnoop* ensures that a crashed host belongs to the $local_{suspect}$ list of

all alive hosts (and which remain alive) at most two cycles after crashing. **By satisfying this condition, the synchronous view agreement, precisely the merge, guarantees that the crashed host gets excluded at most one cycle later, i.e., by the third cycle.**

Detecting crashes in real-time relies on the n_i heartbeats sent by every host in every cycle. Initially the $local_{suspect}$ list of host h only contains h. A host h always suspects itself. At the end of a cycle, every host updates its $local_{suspect}$ list based on the non-excluded hosts it hears from during that cycle. For example, if h_i did not receive any message from h_j (which is part of h_i's current view), then h_i places h_j in the $local_{suspect}$ list. Note that placing h_j in the suspected list does not mean that h_j is excluded from h_i's view; excluding hosts is governed by the synchronous view agreement.

h_j, thus, gets **suspected** (not excluded), at most two cycles after crashing, by all hosts that remain alive (since h_j stops sending heartbeats after crashing and might send a heartbeat and directly crash). The merge of Section IV-A1, **excludes** h_j at most one cycle later, i.e., by the third cycle.

B. Tolerating $s_c \geq 3$ Stale Control Cycles

To tolerate $s_c \geq 3$ cycles, a host h_i needs an array variable (with one entry per host), $count_{stale}$.

The only modification to *ViewSnoop* is induced in the synchronous view agreement part, precisely the merge operation. Hosts now perform *merge* as follows: h_j is declared failed by h_i, according to the description below (otherwise h_j is declared alive).

```
For every h_j
IF (count_stale(h_j)=s_c) DO
        Declare h_j failed
ELSE
        IF (cond1 && cond2) DO
        // cond1: h_j belongs to local_suspect of h_i.
        // cond2: h_j is in the suspected list of all heartbeats received by h_i.
             count_stale(h_j) ++;
        ELSE
             count_stale(h_j) = 1;
        ENDIF
ENDIF
```

C. Approximating ViewSnoop's Probabilistic Guarantees

ViewSnoop satisfies properties SYMS 1 SYMS 3 and SYMS 5 (details are found in Appendix A). We determine, in what follows, p_{agree} and $p_{accurate}$ with which *ViewSnoop* satisfies properties SYMS 2 and SYMS 4. We approximate p_{agree} and $p_{accurate}$ in crash-free executions, i.e., only considering false suspicions resulting from message losses, since we do not assume any particular probability for host crashes (we show in Section VII that crashed hosts are excluded by all alive hosts using *ViewSnoop* in at most 3 cycles). We defer all proofs in this section to Appendix A.

Lemma 2. *The probability, p_{agree}, that all alive hosts in ViewSnoop agree on the view to be installed at a round can be approximated by $p_{agree} = 1 - p_{disagree}$, where:* $p_{disagree} = \sum_{k=1}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{k} [Prob(disagree_A)]^k [1 - Prob(disagree_A)]^{|\mathcal{C}|-k}$, $Prob(disagree_A) = [Prob(1|\pi_A) + Prob(2|\pi_A)]\, P(\pi_A)$,

$$P(\pi_A) = \sum_{|\mathcal{C}|-|\pi_A|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_A|} (1-p)^{n_i(|\mathcal{C}|-|\pi_A|)} \left[1-(1-p)^{n_i}\right]^{|\pi_A|},$$

$$Prob(1|\pi_A) = (1-p)^{n_i \times |\pi_A|} \times \sum_{h=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{h} \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^h \times \left[(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-h-1},$$

$$Prob(2|\pi_A) = [1-(1-p)^{n_i \times \pi_A}] \times \sum_{k=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{k} \left[(1-p)^{n_i(|\pi_A|+1)}\right]^k \times \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-k-1},$$

such that π_A is the the set of hosts which do not have some host A in their $local_{suspect}$ list at the beginning of cycle r and $\mathcal{C}$ is the set of alive hosts in $r-1$, assuming that no host has excluded any other host.

Lemma 3. *The probability $p_{accurate}$ of a host excluded in ViewSnoop to have actually failed can be approximated by $1 - [Prob(1|\pi_B) + Prob(2|\pi_B) + Prob(3|\pi_B)]\, P(\pi_B)$, where:*

$$Prob(1|\pi_B) = (1-p)^{n_i \times \pi_B},$$

$$Prob(2|\pi_B) = \sum_{r=1}^{|\mathcal{C}|-|\pi_B|-1} \binom{|\mathcal{C}|-|\pi_B|-1}{r} \left[(1-p)^{(|\pi_B|+1)n_i}\right]^r \times \left[1-(1-p)^{(|\pi_B|+1)n_i}\right]^{|\mathcal{C}|-|\pi_B|-r-1},$$

$$Prob(3|\pi_B) = \sum_{s=1}^{|\pi_B|} \binom{|\pi_B|}{s} \left[(1-p)^{(|\pi_B|)n_i}\right]^s \times \left[1-(1-p)^{|\pi_B|n_i}\right]^{|\pi_B|-s},$$

$$P(\pi_B) = \sum_{|\mathcal{C}|-|\pi_B|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_B|} (1-p)^{n_i(|\mathcal{C}|-|\pi_B|)} \left[1-(1-p)^{n_i}\right]^{|\pi_B|},$$

such π_B is the set of hosts that heard from some host B in cycle $r-1$ and $\mathcal{C}$ is the set of all alive hosts in cycle r (assuming no exclusions at all hosts).

V. Analytic Evaluation of *ViewSnoop*

Probabilistic performance metrics, like p_{agree} and $p_{accurate}$, cannot be evaluated accurately via experimentation and are best measured analytically. We hence conduct extensive theoretical analyses and simulations addressing: (a) *ViewSnoop*' dependability (b) the effect of network load on the dependability of *ViewSnoop* and (c) *ViewSnoop*'s capability of differentiating between network and host failures.

We address the above points of *ViewSnoop*'s performance and compare with membership schemes based on classic heartbeats. First, we describe the different classic heartbeat-based schemes with which we compare and compute their probabilistic guarantees.

a) Simple fault-exclusion mechanism (SFTM). This mechanism is employed as the basis of most existing membership protocols with real-time guarantees [7], [8], [12], [13]. These existing protocols typically augment SFTM with additional help not supported in the context of this paper, such as allowing hosts to be killed and using additional hardware (see Section VIII). In SFTM, every host broadcasts a heartbeat in every cycle and at the end of a cycle, suspects and excludes all hosts from which it did not hear. A heartbeat here represents an "I am alive" message. Let us denote by p_{agree}(SFTM) and $p_{accurate}$(SFTM), the probability of installing the same view by all alive hosts ($\mathcal{C}$) in some cycle and the probability of a host declared as failed to have actually crashed respectively, given that a message loss probability is $1-p$. Then $p_{agree}(\text{SFTM}) = p^{|\mathcal{C}|(|\mathcal{C}|-1)}$ and

$$p_{accurate}(\text{SFTM}) = 1 - \textstyle\sum_{r=1}^{|\mathcal{C}|-1} \binom{|\mathcal{C}|-1}{r} (1-p)^r p^{|\mathcal{C}|-1-r}.$$

b) M-SFTM. A variant of SFTM where every host broadcasts n_i heartbeats per cycle, as opposed sending a single heartbeat. Sending more heartbeats makes M-SFTM more robust than SFTM to message losses in ways supported by the DCS context discussed in this paper. At the end of a cycle, every host in M-SFTM suspects and excludes all hosts from which it did not hear any heartbeat. A heartbeat has the same structure as in SFTM. Alive hosts in M-SFTM install the same views if each host hears some heartbeat from every other alive host. Hence, $p_{agree}(\text{M-SFTM}) = [1-(1-p)^{n_i}]^{|\mathcal{C}|(|\mathcal{C}|-1)}$. A host h is falsely excluded by at least one other alive host if at least one other host receives none of host h's heartbeats. Thus: $1 - p_{accurate}(\text{M-SFTM}) = \sum_{k=1}^{|\mathcal{C}|-1} \binom{|\mathcal{C}|-1}{k} [(1-p)^{n_i}]^k [(1-(1-p)^{n_i})]^{|\mathcal{C}|-1-k}$.

c) Ring Algorithm. A variant of SFTM, this time not using all-to-all communication, inspired from Larrea et al. [27]–[29]. Hosts send heartbeats following a ring structure. Initially host 1 sends heartbeats only to host 2, host 2 to host 3, ..., and host n to host 1 (n : total number of hosts). Similar to Section IV, we describe the algorithm for $s_c = 3$.

The ring algorithm uses a boolean variable, "$suspect$", initially set to false.

In every cycle, host$_i$ sends a heartbeat tagged with the cycle number, n_i times to host$_{i+1}$ following it on the ring of the current system view. The heartbeat has the same structure as in SFTM. At the end of the control cycle, every host$_i$ checks if it received some heartbeat from host$_{i-1}$ of the current system view. If no such message is received, then host$_i$ updates its $suspect$ variable to true (false otherwise).
At the beginning of cycle r, every host$_i$ executes:

```
Install V = (r, M) such that M = M', where (r − 1, M')
is the view of the previous control cycle.
IF (suspect) DO
   Broadcast < id(host_{i−1}), crash > message n_i times.
   Declare host_{i−1} failed & install at the beginning of
           cycle r + 1, V = (r + 1, M''): host_{i−1} ∉ M''.
   Set suspect = false
   Listen in control cycle r + 1 to heartbeats of
           host_{i−1} in the new V = (r + 1, M).
ELSE
   Install at beginning of cycle r + 1, V = (r + 1, M), M
   being the system view at cycle r.
ENDIF
At the end of a control cycle r, every host_i executes:
IF (< id(host_x), crash > is received) DO
       Declare host_x failed and install at beginning of
cycle r + 1, V = (r + 1, M'') such that host_{i−1} ∉ M''.
ENDIF
```

The probability that hosts are in agreement (details are deferred to Appendix D) is: $p_{agree}(\text{Ring}) = [1 - (1-p)^{n_i}]^{|\mathcal{C}|}$.

The probability that a correct host is falsely suspected and excluded in a cycle, i.e., the probability that a host is placed in the suspected list of at least one other host in some cycle is:

$$1-p_{accurate}(\text{Ring}) = (1-p)^{n_i} + (1-p)^{n_i} \sum_{h=1}^{|\mathcal{C}|-1} \binom{|\mathcal{C}|-1}{h} [(1-p)^{n_i}]^h [1-(1-p)^{n_i}]^{|\mathcal{C}|-1-h}.$$

Now we evaluate the dependability of *ViewSnoop* compared to the aforementioned classic heartbeat-based schemes.

A. ViewSnoop's Dependability

We evaluate *ViewSnoop*'s dependability by comparing the values of p_{agree} and $p_{accurate}$ achieved by *ViewSnoop* versus those obtained using SFTM, M-SFTM and the ring algorithms. Any gain in p_{agree} and/or $p_{accurate}$ translates into a more dependable SYMS implementation and a better DCS availability[4]. We simulate the values of p_{agree} and $p_{accurate}$ for all algorithms using $n_i \in \{1, 2, 4, 8\}$ broadcast messages per control cycle. Note that for $n_i = 1$, SFTM and M-SFTM become the same algorithm. Simulations are run for $p \in \{0.8, 0.9, 0.99, 0.999, 0.9999, 0.99999, 0.999999\}$ and $\mathcal{C} \in \{3, 10, 100, 1000\}$, where p is the probability of a host receiving a broadcast message successfully and $\mathcal{C}$ is the number of alive hosts in the system. Due to space limitations, we report our results, in Figure 4, for $\mathcal{C} \in \{10, 100\}$, additional values for $\mathcal{C} \in \{3, 1000\}$ can be found in Appendix B.

1) Gain in Agreement Probability. Figure 4A (a),(b) and (e),(f) report respectively the ratios of p_{agree} obtained by *ViewSnoop* w.r.t. that obtained by M-SFTM (SFTM) and ring.

The following remarks are in order:

1. *ViewSnoop* has a higher agreement probability compared to all other classic heartbeat-based mechanisms and under all settings, i.e., for all values of p, n_i and $\mathcal{C}$.
2. For a fixed value of p and a fixed n_i, the gain in the agreement probability for *ViewSnoop* over all other classic heartbeat-based mechanisms increases exponentially as the number of alive hosts $\mathcal{C}$ increases.
3. Given a fixed number of alive hosts ($\mathcal{C}$), the positive gain in the agreement probability of *ViewSnoop* compared to all other mechanisms tends asymptotically to zero (i.e., no gain) as $p \to 1$ and as n_i increases (when messages losses are fully masked all algorithms provide the same guarantees).

2) Gain in Accuracy. The gain in the probabilistic accuracy of *ViewSnoop* over that of all classic heartbeat-based mechanisms can be observed in Figure 4A (c),(d) and (g),(h). The probability of falsely excluding an alive process achieved by all classic heartbeat-based mechanisms is lower bounded by the probability achieved by *ViewSnoop*, allowing it to have the best accuracy. This lower bound becomes tighter as $p \to 1$ and $n_i \to \infty$ while it becomes more relaxed as $\mathcal{C} \to \infty$.

Conclusion. ViewSnoop indeed offers a more dependable service, compared to SFTM, M-SFTM and the ring algorithm, enhancing thus the availability of a DCS. The significance of this improvement relies on: (i) the number of heartbeats sent every cycle, (ii) the reliability of the communication and (ii) the size of the DCS. For DCSs suffering from communication losses, our algorithm provides superior probabilistic guarantees for critical cyclic control applications (where the number of sent heartbeats per cycle is scarce) compared to mechanisms based on sending simple heartbeats. In fact, we show in Appendix C that both probabilities with which *ViewSnoop* implements SYMS properties, i.e., p_{agree} and $p_{accurate}$, tend to 1 as the number of hosts in the system tends to ∞.

B. The Effect of Network Load

We study, at a finer granularity, the effect of network load on the guarantees of algorithms implementing SYMS. Network load can be split into: (i) message complexity, i.e., the number of heartbeat messages sent per host per cycle, and (ii) message size. We study the impact of these factors individually.

[4]Lower accuracy means more correct hosts get excluded, increasing the risk of downtime due to the lack of processing resources.

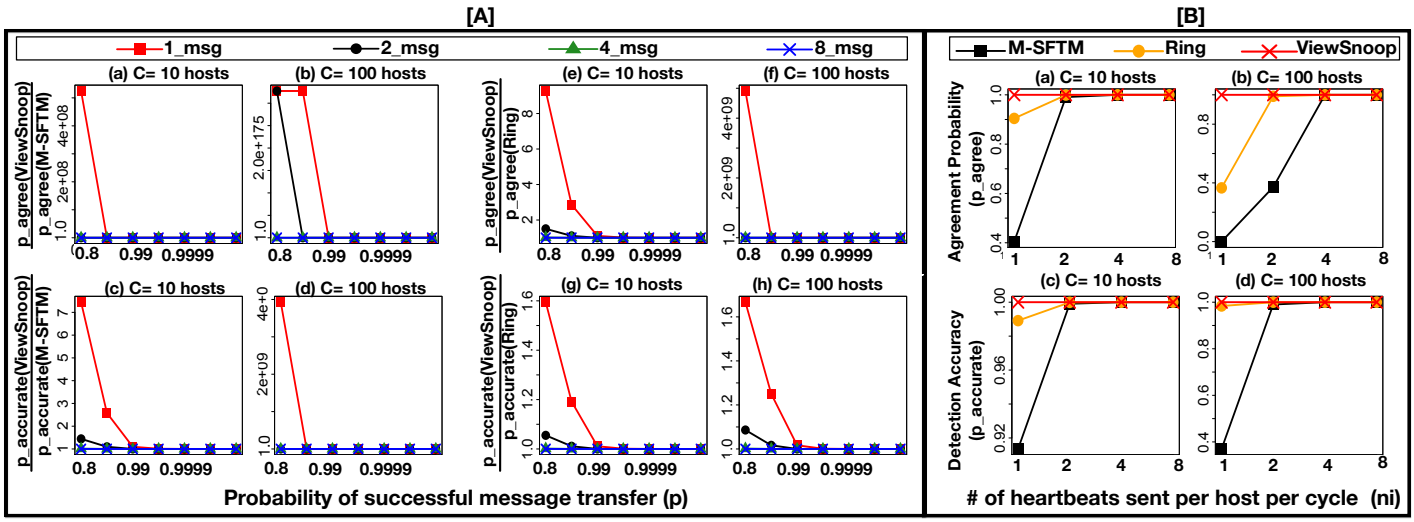

Figure 4. A. The ratio of the agreement probability and accuracy of ViewSnoop w.r.t. to other algorithms; B. Values of p_{agree} and $p_{accurate}$ of all algorithms versus varying values of n_i for $p = 0.99$.

1) Message complexity. The table below summarizes the message complexitiesin the absence of host crashes and losses.

Algorithm	Message Complexity	Message Content
SFTM	$O(\mathcal{C}^2)$	2 integers
M-SFTM	$O(n_i\mathcal{C}^2)$	2 integers
Ring	$O(n_i\mathcal{C})$	2 integers
ViewSnoop	$O(n_i\mathcal{C}^2)$	2 integers +1 string

Observation 1. *Allowing hosts to send more heartbeats per cycle improves an algorithm's probabilistic guarantees.*

Our results in Figure 4B show that both, p_{agree} and $p_{accurate}$, for all algorithms, i.e., *ViewSnoop*, M-SFTM and ring, increase as n_i (the number of heartbeats sent per host per cycle) increases. This is expected as sending the same message multiple times helps mask potential losses of that message.

Observation 2. *Consider algorithms A and B with message complexities $O(A)$ and $O(B)$ respectively. If $O(A) \geq O(B)$ then the statement: "B is at most as good as A", w.r.t. the ensured reliability and availability,* ***does not*** *hold.*

Our results in Figure 4B show that the ring algorithm (with lower message complexity) provides better guarantees than M-SFTM for $p = 0.99$, both in terms of p_{agree} and $p_{accurate}$. This result also holds for various values of p (see Appendix B).

Conclusion. Increasing the number of heartbeats sent per host per cycle of an algorithm implementing SYMS increases the probabilistic guarantees of that algorithm; however this relation does not hold across algorithms. In other words, the performance of distinct algorithms cannot be compared solely based on their message complexity.

2) Message Size and Structure. We investigate whether *ViewSnoop* benefits from sending more information in a heartbeat than simply: "I am alive", to achieve better guarantees. We compare M-SFTM and *ViewSnoop*, as they have the same message complexity but differ in message size. Such a comparison shows the impact of exchanging local views versus simple heartbeats. Figure 4B shows a positive improvement in p_{agree} and in $p_{accurate}$ when local views are exchanged. The improvement over M-SFTM, for 10 hosts, is about $9.2\times$, increasing exponentially with the increasing system size.

Conclusion. Appending local views to heartbeats allows *ViewSnoop* to increase its probabilistic guarantees and thus the availability of the DCS. The improvement is most significant in large DCSs running critical applications (small n_i).

C. Distinguishing Host Crashes from Message Losses

Distinguishing host crashes from message losses is very important in DCSs. In case of message losses where hosts are still alive, a DCS can benefit from this information to update the configurations such that tasks requiring communication would not be allocated to hosts connected by bad links or different routes for communication are used instead. In all classic heartbeat-based mechanisms, roughly speaking, a host A knows the state of host B (if B is alive or not) only if A and B can communicate, even with the use of sequence numbers [24]. As long as communication between A and B is down then A has no idea about B's state. In contrast, *ViewSnoop*, by exchanging views, allows A to know about the state of B from other hosts even if communication between A and B is down. *ViewSnoop* can detect the failure of communication, when for example host B is in the suspect list of host A while A still sees that host B is not in the suspect list of all other hosts that A hears from. Let p_{com_fail} be the probability that a host A detects correctly the communication failure between itself and another host. Note that all other classic heartbeat-based mechanisms are incapable of detecting correctly a communication failure without a trade-off in $p_{accurate}$, while *ViewSnoop* can do it with probability:

$$p_{com\_fail} = 1 - (1-p)^{n_i(|\mathcal{C}|-1)} - \sum_{k=1}^{|\mathcal{C}|-1} \binom{|\mathcal{C}|-1}{k} [(1-p)^{n_i}]^k [1-(1-p)^{n_i}]^{|\mathcal{C}|-1-k} \times \sum_{h=1}^{|\mathcal{C}|-1-k} \binom{|\mathcal{C}|-1}{h} [(1-p)^{n_i}]^h [1-(1-p)^{n_i}]^{|\mathcal{C}|-1-k-h}$$

Notice that p_{com_fail} tends to 1 as n_i and $|\mathcal{C}|$ increase. This means that *ViewSnoop* can detect communication failures with high probability in large systems where hosts are not limited in the number of heartbeats they can send per control cycle.

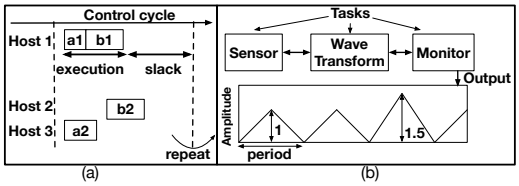

Figure 5. (a) A FASA control cycle; (b) *Waveform*: an example application.

VI. Run-time Environment, Application and Implementation Details

A. Run-time Environment

We implemented *ViewSnoop* and deployed it in FASA [7], an industrial automated DCS framework, whose behavior adheres to the description in Section II. A cycle in FASA consists of a phase called the "execution period", followed by a phase called the "slack period" (see Figure 5 (a)). The execution period is the time a host utilizes for executing tasks assigned to it. The slack period is the remaining time of the cycle (used for running background operations if any is needed). The scheduler in FASA computes global configurations statically and installs the configurations relative to the alive hosts. The scheduler, based on the configuration, knows which hosts need to communicate with each other. Accordingly, the FASA scheduler builds abstract communication *channels* between communicating hosts on top of the unidirectional links. These channels can be configured to use different underlying links.

For our experiments, we deploy *ViewSnoop* in a FASA system where the scheduler can accommodate a maximum of one failure. The scheduler embodies pre-computed configurations to re-distribute application tasks on hosts, when only a single host can crash. The failure of more than one host would cause the system to stop executing.

B. Application

We execute on FASA a cyclic control application called *Waveform* (see Figure 5 (b)). *Waveform* is a simplified example of an industrial control application (extracted from a real setting) that reads some input variable, performs calculations (e.g., a cascaded feedback loop), and finally writes some output to a field-bus I/O interface. The application is executed every cycle, by periodically executing the application's tasks. In the example, a new input value is provided by the *Sensor* task at the beginning of each cycle. The input follows a triangular periodical signal. The *WaveTransform* task performs certain calculations that change the input signal. Specifically, the *WaveTransform* task in this example observes the input signal to learn its amplitude, base, and period and increases the upper half of the triangular wave amplitude by a factor of 1.5 every third period. This output is fed into a *Monitor* task, which prepares the value for output to a field-bus I/O interface.

C. ViewSnoop Implementation

ViewSnoop is implemented within the FASA distributed scheduler (see Figure 6) in C++. For communication, *ViewSnoop* has access to a UDP broadcast primitive (without acknowledgments and prone to communication failures).

At the beginning of every cycle, the *ViewSnoop* module on every host broadcasts a heartbeat message. In our current implementation, *ViewSnoop* sends a single heartbeat per cycle, i.e., $n_i = 1$. This scheme could be extended to multiple broadcast messages, however, taking into account the cycle time (in Section V we evaluate *ViewSnoop* with $n_i \geq 1$).

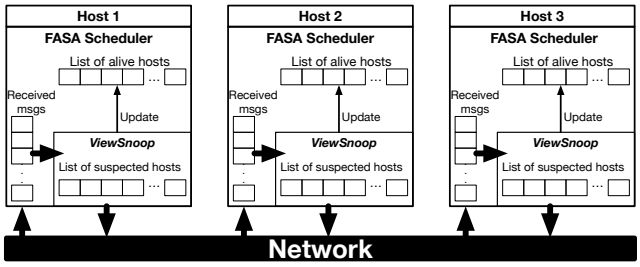

Figure 6. Architecture of *ViewSnoop* within FASA.

The *ViewSnoop* module on each host maintains a local list of suspected hosts (see Figure 6). This list is implemented as a vector containing the $host\_ids$ of suspected hosts. The heartbeat message in *ViewSnoop* is implemented as an object encapsulating the control cycle number and the list of suspected hosts. For programming simplicity, this object is parsed into a string when transmitted on the network. During the slack period, the *ViewSnoop* module on h_i checks for the *current_host_ids*: the ids of the hosts from which h_i received heartbeat messages for the current cycle. *ViewSnoop* decides based on its local list of suspected processes and the *current_host_ids* to update the list of alive hosts observable by the scheduler. Upon observing a change in the list of alive hosts, the scheduler activates the corresponding configuration.

VII. Experimental Evaluation of *ViewSnoop*

We evaluate experimentally the performance and cost of *ViewSnoop* addressing the following points: (a) the time for detecting and excluding crashed hosts, (b) the time the system remains available in the presence of communication losses, given no host failures, (c) the overhead *ViewSnoop* adds to a DCS and (d) how fast *ViewSnoop* accommodates host recoveries. We compare the performance of *ViewSnoop* with that of SFTM, the mechanism employed in most existing membership protocols with real-time guarantees (described Section V). Whenever needed, we inject message losses in the network at the receiver side; we assign a fixed *success probability* p with which a sent message is successfully received by a destination host (unreliable broadcast). This is besides any other message losses that can happen in the network; these losses can be correlated and can result from collisions, contention, etc, since we conduct our evaluation in a real production DCS.

Hardware Description. We deploy *ViewSnoop* within FASA [7] and precisely in the same industrial setting in which FASA was originally implemented, tested and run [7]. That is, three Mac Minis with dual-core Intel i7-2620M @2,7GHz CPU, 4 GB RAM, and Gigabit Ethernet network connection (a similar implementation setting was also used for example in the RTCAST real-time membership protocol [12]). Our implementation on three machines is used to validate and estimate the overhead of *ViewSnoop* versus that of SFTM (also deployed in FASA [7]). Performance on larger DCSs is rigorously simulated in Section V.

The machines are given unique ids $(1, 2$ and $3)$ and we refer to them as hosts. All machines run Ubuntu 12.10, Kernel: $3.5.0-24$x86_64. The hosts use control cycles of 5 ms and are synchronized using PTP [30]. The cycle duration in practice varies according to the applications, e.g., 8 ms to 10 ms for substation automation and low-level robot interfaces [31] and up to 1 s for less critical temperature-drive applications.

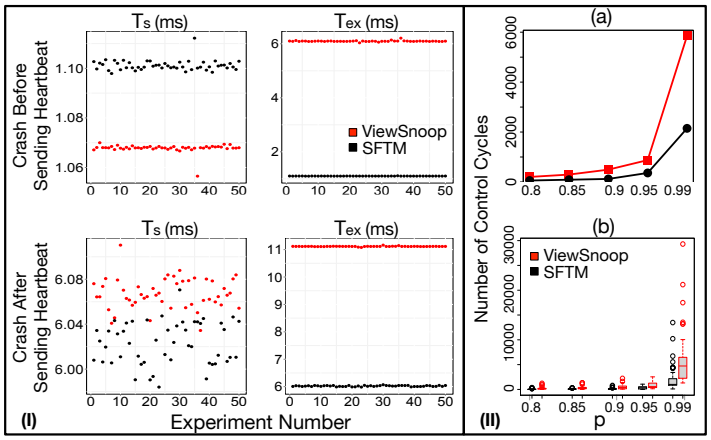

Figure 7. I. Time to suspect (T_s) and exclude (T_{ex}) crashed hosts; II. Number of control cycles until an alive host is excluded (the higher the better): (a) mean; (b) distribution.

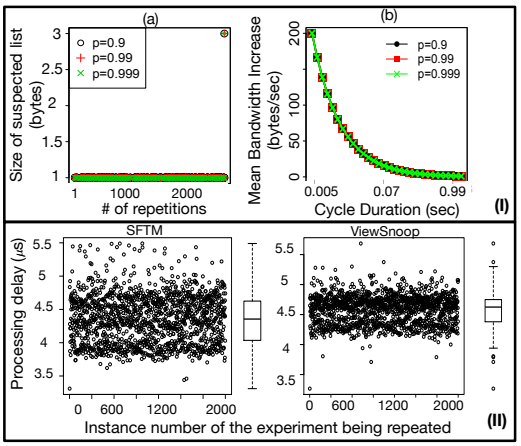

Figure 8. I. Network overhead: (a) 1 out of 3000 heartbeats contained a 3 byte instead of 1 byte suspicion list; (b) *ViewSnoop*'s mean bandwidth increase versus cycle duration; II. Processing delay of SFTM and *ViewSnoop* in μs.

A. Time to Exclude Crashed Hosts

We verify experimentally our theoretical claims (Section IV) regarding *ViewSnoop*'s speed of excluding crashed hosts, and compare them to those achieved by SFTM. To this end, we crash host 1 at the 50^{th} cycle in two manners: (i) before sending a heartbeat in the 50^{th} cycle and (ii) after sending that heartbeat. We measure the following:

1. The time span until some host in the system suspects host 1 after it fails (T_s).
2. The time span until host 1 gets excluded after its failure by all hosts in the system (T_{ex}).

Theoretically, $T_s \leq 2$ and $T_{ex} \leq 3$ cycles in *ViewSnoop* (see Section IV) and $T_s = T_{ex} \leq 2$ in SFTM (implied from SFTM's description earlier this section). We repeat the experiment 50 times and report the values in Figure 7-I. Our results show that when host 1 crashes before sending a heartbeat, host 1 gets suspected, in *ViewSnoop*, after ≈ 1 ms, i.e., at the same cycle it crashed in (the 50^{th} cycle) and gets excluded from the system after 6 ms (at the 51^{st} cycle). When host 1 crashes after sending a heartbeat, host 1 gets suspected, in *ViewSnoop*, 8 ms after crashing (at the 51^{st} cycle) and excluded from the system 11 ms after crashing (at the 52^{nd} cycle). Our results also show that, in SFTM, $T_s = T_{ex}$ and that host 1 is suspected at the same cycle as in *ViewSnoop* but excluded one cycle earlier than in *ViewSnoop*.

B. Mean Time to Failure

We assess the system's reliability by measuring the mean time to failure of *ViewSnoop*, focusing here on violating P4 of the group membership properties. To this end, we count the number of control cycles in a crash-free execution until an alive host is excluded by *ViewSnoop* by mistake. We consider crash-free executions (we do not crash any host) and simulate message losses on the network by specifically having a receiver of a broadcast heartbeat deliver that message randomly with fixed probability $p \in \{0.8, 0.85, 0.9, 0.95, 0.99\}$; the message is dropped otherwise. We do not consider values of $p > 0.99$ in experimentation since it requires weeks or even months to obtain the desired numbers. We account, however, for such values of p in our analytic evaluation (Section V).

For each value of p, we run the system for 50 times, measuring in each how long it takes the system to declare a correct host as failed. We plot the average values and the detailed distribution in Figure 7-II (a) and (b) respectively. Our results show that *ViewSnoop* can keep a correct host in the system for a much longer time than SFTM. This means that a system with *ViewSnoop* is expected to have a higher availability (processing resources are available for longer times) and reliability (falsely suspects processes at a lower rate) than with SFTM. We also consider a variant of SFTM: a host is excluded if no heartbeat is received from that host for two consecutive cycles. This variant is considered to compare *ViewSnoop* and SFTM when having the same speed of excluding crashed hosts. Even with this variant, *ViewSnoop* amounts to a better reliability than SFTM. The probability of excluding a correct host in *ViewSnoop* is $[(1-p)^2(1-p^2)]$ versus $(1-p)^2$ in SFTM. Correct hosts are thus expected to stay included in the system using *ViewSnoop* for a longer duration. The reason is that *ViewSnoop* allows hearing about hosts from other alive ones.

C. Costs of ViewSnoop on a DCS

1) Network cost. We quantify the network overhead of *ViewSnoop* by measuring the additional bandwidth required by our *ViewSnoop* implementation in comparison to that needed by SFTM. *ViewSnoop* requires every host to append to the heartbeat of SFTM, a string containing the ids of the suspected hosts. Such a heartbeat is sent every control cycle, in this case every 5 ms. As in Section VII-B, we introduce message losses; messages are successfully received with probability $p \in \{0.9, 0.99, 0.999\}$. We run the system until it halts and record the size (in bytes) of all suspected lists appended to heartbeats in that run by all hosts. For each value of p, we repeat this experiment $\approx$ 20 times ($\approx$3,000 heartbeats). We report the values in Figure 8-I(a). Our results show that the size of the suspicion lists is consistent in all repetitions: 1 out of 3000 heartbeats contained a 3 byte instead of 1 byte suspicion list. 3 bytes lists are observed at the end of the experiments, since we run the system until one host is falsely detected as failed, after which all hosts stop operation. This causes all hosts to suspect each other increasing the size of the list. These values do not vary between the different values of p. An Ethernet packet in SFTM is 62 bytes (IPv4) and 82 bytes (IPv6), meaning that our algorithm induces, on average, an overhead of 1.6% and 1.3% respectively, compared to SFTM.

We also plot, in Figure 8-I(b) the average additional bandwidth of *ViewSnoop* (compared to SFTM) as a function of the cycle duration (varying from 5 ms to 1 s). The additional bandwidth for *ViewSnoop* is 200 bytes/sec for 5 ms cycles and decreases exponentially as the cycle duration increases.

2) Processing cost. Under the same experimental setup as for evaluating network costs, we measure *ViewSnoop*'s processing cost, i.e., how much delay does *ViewSnoop* add to the regular processing time of hosts. To that end, we measure during a control cycle the time a host spends to check for crashed hosts, suspect hosts, update the list of suspected hosts and update the content of the heartbeat message to be sent. The statistics are consistent for the different values of p, so for brevity we report statistics for $p = 0.99$. Figure 8-II shows that, on average, *ViewSnoop* requires 0.3 μs more processing time per cycle compared to SFTM. More importantly, such processing is done in the slack period of a cycle, which typically is 20% of the cycle duration, i.e., 1 ms for a 5 ms cycle. Our algorithm thus does not delay any application as it consumes on average 0.46% of the slack period, which is entirely dedicated for background operations by design. We also observe that processing delays above 5 μs occur fewer times, attributed to cases when hosts are falsely suspected.

D. Host Recoveries

For simplicity, we have discussed in the main paper SYMS and *ViewSnoop*, our implementation of SYMS, without considering host recoveries. In fact, both SYMS and *ViewSnoop* can be easily extended to encompass recoveries. In this recovery setting, the maximum number of hosts in the system is known ahead of time. However, since hosts can be excluded from the system view, due to actual crashes or communication faults, these hosts can try to enter the system again. Let $\mathcal{W}(r)$ be the set of non-crashed hosts at the beginning of cycle r (a non-crashed host does not have to be part of the system view). For coherence, an "alive" host here is a host that has not been declared yet as crashed by any host. A recovered host B becomes "alive" only starting from the cycle in which B's view includes all alive hosts and the views of all alive hosts include B. To allow recoveries, SYMS 1 is updated to: if a host installs a view $V = (id, M)$ and then $V' = (id' = r, M')$, then $id < r$ and $\{M' - \{M \cap M'\}\} \subseteq \mathcal{W}(r)$. Accordingly, *ViewSnoop* is adapted to account for recoveries as follows:

1) A recovered host B, wanting to join the system, broadcasts heartbeats at the beginning of every cycle. Initially, B only has itself in its view. B learns about the alive hosts in the system and includes them in its view according to (2) below.

2) If a host A receives a heartbeat from host B in cycle r (where B is not in A's view), then A removes B from its suspected list. A includes B in its view in cycle r' if A does not suspect B and all hosts that A heard from in cycle r' do not suspect B.

We implement this crash recovery extension of *ViewSnoop* and evaluate how long it takes a recovering host to be included in the view of all other alive hosts. A broadcast heartbeat is delivered by a host with probability $p \in \{0.8, 0.85, 0.9, 0.99\}$. We force host 1 to stop sending heartbeats at the 5^{th} cycle and start sending heartbeats again at the 11^{th} cycle. This behavior

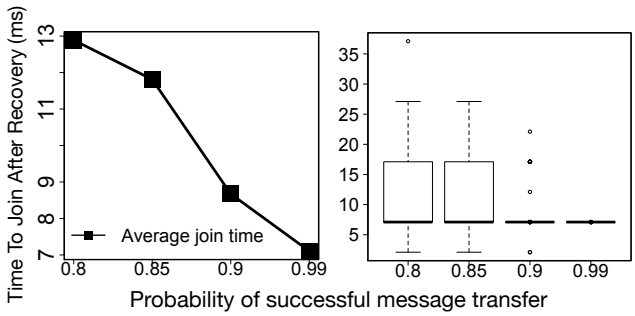

Figure 9. Time for recovered hosts to join the system in *ViewSnoop*.

ensures that host 1 is declared failed by both host 2 and host 3, before trying to join the system back at the 11^{th} cycle. We measure $T_{recover}$, the time from the beginning of the 11th control cycle until host 1 is included in the view of both host 2 and host 3. For each value of p we repeat the experiment 50 times and report our results in Figure 8-III. We notice that as p increases, i.e., less losses, the time for a new host to join the system approaches two control cycles. This duration is an upper bound on the time a process needs to re-join the system in the absence of message losses. We can see for $p \in \{0.8, 0.85, 0.9\}$ cases where a host can re-join in one cycle. This happens when host 2 and host 3 receive host 1's heartbeat at the 11^{th} cycle, but do not hear from each other (such fast re-joins can only happen in unreliable communication).

In comparison, in SFTM host 2 and host 3 include host 1 in their views as soon as they receive a heartbeat from host 1. Host 1, thus, joins the system in the same cycle with probability p^2, after one cycle with probability $(1-p^2)p^2$, and after s cycles with probability $(1-p^2)^{(s-1)}p^2$. In the presence of losses, host 1 joins the system, on average, after

$$\frac{1-p^2}{p^2}$$

cycles. Given no message losses, a recovered host, in SFTM, is recognized by all alive hosts in the system in less than one cycle (one cycle less than *ViewSnoop*).

VIII. Related Work

Membership services have been addressed in different contexts [3], [4], [32]–[43]. For example, detecting host crashes in real-time while guaranteeing the desired quality of service needed by applications has been targeted in [44]–[51]. However, none of these works addressed membership issues, precisely the issue of providing a consistent view of failures.

So in what follows, we focus on existing work on membership services in real-time context.

Kopetz and Grünsteidl [8] proposed the time-triggered protocol (TTP) for distributed real-time control applications. TTP provides many services including membership. TTP assumes time division multiple access (TDMA)[5] as means to organize sender transmission. A node is considered failed when no message is received from that node in its designated transmission slot. Also, a sender node itself can decide if it is not operating correctly, and accordingly crash itself, based

[5]TDMA divides the medium access into slots such that in each TDMA round nodes transmit a fixed amount of traffic in the preallocated slots.

on: (i) internal detection mechanisms, (ii) acknowledgments received relative to a window of previous transmissions and (iii) frame rejections (by preforming a specific CRC check). Disagreement is resolved, with high probability, by excluding nodes that do not agree with the majority. Barbosa et al. [13] devised a protocol using TDMA, where each node must acknowledge messages from k other nodes in the membership group. Membership agreement is guaranteed if $f \leq k-1$ failures occur during n consecutive transmission slots (n being the total number of nodes in the system). Rufino et al. [23] proposed failure detection and membership services to enhance the dependability of distributed systems interconnected by field-bus technologies, namely CAN, to levels comparable to TTP-based systems. The major component in their technique is the CAN enhanced layer: a combination of the CAN standard layer with some simple machinery and low-level protocols. Abdelzaher et al. [12] proposed *RTCast*, a multicast and membership service for real-time process groups sending periodic or aperiodic messages. *RTCast* relies on a ring topology in which processes take turn in sending heartbeats. *RTCast* relies on processes being able to crash themselves, namely when detecting receive omissions. Amir et al. proposed Totem [15], a reliable total ordered broadcast protocol assuming a ring topology. The Totem protocol depends on acknowledgments and retransmission mechanisms to overcome communication faults and can provide real-time message delivery with high probability. Clegg and Marzullo [16] designed group membership with low overhead, low cost of handling failures and supporting simple schedulability analysis. The proposed solution abstains from sending heartbeats but rather relies on other layers to exchange enough messages to ensure failure accuracy and liveness of membership agreement.

The work in this paper differs essentially at two levels: (i) required guarantees and (ii) implementation. On the level of guarantees, we require crashes to be synchronously detected on all alive hosts and within bounded periods from the crash. In contrast with existing work, disagreement, even for few rounds after which agreement maybe achieved, must be avoided. We thus define a probabilistic notion of agreement which we seek to maximize. Implementation wise, our protocol (*ViewSnoop*) relies on the periodic exchange of messages, however not in a TDMA fashion as opposed to [8], [13] enabling *ViewSnoop* to tolerate message collisions. Also, as opposed to [23], we do not assume a specific type of interconnection between hosts (i.e., field-bus), we just consider communication is prone to losses, not relying on additional hardware or acknowledgments. Processes in *ViewSnoop* do not kill themselves, as opposed to [8], [12], where the DCS's computing resources can be easily depleted, jeopardizing the system's availability. To this end, we employ a technique not explored in any of the previous work. Our approach relies on exchanging local views as opposed to exchanging classic heartbeats (as [8], [12], [13], [15]) and having self-crashing capabilities.

IX. Concluding Remarks

In this paper, we defined the membership properties required in DCSs. In their implementable form, these properties take the form of a probabilistic real-time membership service called SYMS. We proposed *ViewSnoop*, an algorithm based on exchanging local views between hosts, as a way to implement SYMS with high probability guarantees and low overhead. We evaluated *ViewSnoop* both analytically as well as experimentally via an implementation within FASA, an industrial DCS framework. Our results convey that *ViewSnoop* indeed provides a more dependable service, compared to classic heartbeat mechanisms, thus enhancing a DSC's availability. Additionally, *ViewSnoop* can distinguish host failures from message losses. Schedulers in DCSs can thus compute better configurations, in the sense that tasks requiring information exchange are not allocated to hosts connected by bad links.

References

[1] (2009) Automation services reduce downtime for manufacturers. [Online]. Available: http://www.automationworld.com/automation-services-reduce-downtime-manufacturers

[2] F.-L. Lian, J. Moyne, and D. Tilbury, "Network design consideration for distributed control systems," *IEEE Trans. Control Syst. Technol.*, vol. 10, 2002.

[3] E. Anceaume, A. Fernández, A. Mostefaoui, G. Neiger, and M. Raynal, "A necessary and sufficient condition for transforming limited accuracy failure detectors," *J. Comput. Syst. Sci.*, vol. 68, 2004.

[4] P. Felber, X. Défago, R. Guerraoui, and P. Oser, "Failure detectors as first class objects," in *DOA*, 1999.

[5] G. Lipari, P. Gai, M. Trimarchi, G. Guidi, and P. Ancilotti, "A hierarchical framework for component-based real-time systems," in *LNCS*, 2004, vol. 3054.

[6] G. Lipari, E. Bini, and G. Folher, "A framework for composing real-time schedulers," *ENTCS*, vol. 82, 2003.

[7] M. Oriol, M. Wahler, R. Steiger, S. Stoeter, E. Vardar, H. Koziolek, and A. Kumar, "Fasa: A scalable software framework for distributed control systems," in *ISARCS*, 2012.

[8] H. Kopetz and G. Grunsteidl, "Ttp - a time-triggered protocol for fault-tolerant real-time systems," in *FTCS*, 1993.

[9] E. Latronico and P. Koopman, "Design time reliability analysis of distributed fault tolerance algorithms," in *DSN*, 2005.

[10] P. L. Martnez, L. Barros, and J. M. Drake, "Design of component-based real-time applications," *J. Syst. Softw.*, vol. 86, 2013.

[11] R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, "Right on time distributed shared memory," in *RTSS*, 2016.

[12] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin, "Rtcast: lightweight multicast for real-time process groups," in *RTTAS*, 1996.

[13] R. Barbosa, A. Ferreira, and J. Karlsson, "Implementation of a flexible membership protocol on a real-time ethernet prototype," in *PRDC*, 2007.

[14] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Trans. Comput. Syst.*, vol. 13, 1995.

[15] L. Moser and P. Melliar-Smith, "Probabilistic bounds on message delivery for the totem single-ring protocol," in *RTSS*, 1994.

[16] M. Clegg and K. Marzullo, "A low-cost processor group membership protocol for a hard real-time distributed system," in *RTSS*, 1997.

[17] C. Almeida, "Handling qos in a dynamic real-time environment," in *WORDS*, 2003.

[18] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*. Springer-Verlag New York, 2011.

[19] G. V. Chockler, I. Keidar, and R. Vitenberg, "Group communication specifications: A comprehensive study," *ACM Comput. Surv.*, vol. 33, 2001.

[20] R. Friedman and R. van Renesse, "Strong and weak virtual synchrony in horus," in *SRDS*, 1996.

[21] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *USENIXATC*, 2010.

[22] J. B. Leners, T. Gupta, M. K. Aguilera, and M. Walfish, "Taming uncertainty in distributed systems with help from the network," in *EuroSys*, 2015.

[23] J. Rufino, P. Veríssimo, and G. Arroz, "Node failure detection and membership in canely," in *DSN*, 2003.

[24] I. Soraluze, R. Cortiñas, A. Lafuente, M. Larrea, and F. Freiling, "Communication-efficient failure detection and consensus in omission environments," *Inf. Process. Lett.*, vol. 111, 2011.

[25] D. Dzung, R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, "To transmit now or not to transmit now," in *SRDS*, 2015.

[26] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, 1996.

[27] A. Lafuente, M. Larrea, I. Soraluze, and R. Cortias, "Communication-optimal eventually perfect failure detection in partially synchronous systems," *JCSS*, vol. 81, 2015.

[28] M. Larrea, S. Arevalo, and A. Fernandez, "Efficient algorithms to implement unreliable failure detectors in partially synchronous systems," *LNCS*, vol. 1693, 1999.

[29] M. Larrea, A. F. Anta, and S. Arévalo, "Implementing the weakest failure detector for solving the consensus problem," *IJPEDS*, 2013.

[30] "Ieee standard profile for use of ieee 1588 precision time protocol in power system applications," *IEEE Std C37.238-2011*, July 2011.

[31] M. Felser, "Real-time ethernet - industry prospective," *Proceedings of the IEEE*, vol. 93, 2005.

[32] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo, "Highly available intrusion-tolerant services with proactive-reactive recovery," *TPDS*, vol. 21, 2010.

[33] R. Friedman, A. Mostéfaoui, and M. Raynal, "A weakest failure detector-based asynchronous consensus protocol for f<n," *Inf. Process. Lett.*, vol. 90, 2004.

[34] C. Fetzer, "Perfect failure detection in timed asynchronous systems," *IEEE Trans. Comput.*, vol. 52, no. 2, 2003.

[35] G. Khanna, I. Laguna, F. A. Arshad, and S. Bagchi, "Stateful detection in high throughput distributed systems," in *SRDS*, 2007.

[36] J. Balasubramanian, S. Tambe, C. Lu, A. Gokhale, C. Gill, and D. C. Schmidt, "Adaptive failover for real-time middleware with passive replication," in *RTAS*, 2009.

[37] M. Serafini, P. Bokor, N. Suri, J. Vinter, A. Ademaj, W. Brandstatter, F. Tagliabo, and J. Koch, "Application-level diagnostic and membership protocols for generic time-triggered systems," *TDSC*, vol. 8, 2011.

[38] S. Varadarajan and T. Chiueh, "Automatic fault detection and recovery in real time switched ethernet networks," in *INFOCOM*, vol. 1, 1999.

[39] B. Rong, I. Khalil, and Z. Tari, "An adaptive membership algorithm for application layer multicast," in *ICNS*, 2006.

[40] J. K. Muppala, S. P. Woolet, and K. S. Trivedi, "Real-time systems performance in the presence of failures," *Computer*, vol. 24, 1991.

[41] D. M. Moraes and E. Duarte, "A failure detection service for internet-based multi-as distributed systems," in *ICPADS*, 2011.

[42] F. Greve, P. Sens, L. Arantes, and V. Simon, "A failure detector for wireless networks with unknown membership," in *Euro-Par*, 2011.

[43] A. Fernandez Anta, S. Rajsbaum, and C. Travers, "Brief announcement: Weakest failure detectors via an egg-laying simulation," in *PODC*, 2009.

[44] M. Bertier, O. Marin, and P. Sens, "Performance analysis of a hierarchical failure detector," in *DSN*, June 2003, pp. 635–644.

[45] C. Fetzer, M. Raynal, and F. Tronel, "An adaptive failure detection protocol," in *PRDC*, 2001.

[46] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *DSN*, 2002.

[47] N. Hayashibara, X. Defago, and T. Katayama, "Two-ways adaptive failure detection with the phi-failure detector," in *ICAC*, 2003.

[48] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," in *PODC*, 2001.

[49] L. Falai and A. Bondavalli, "Experimental evaluation of the qos of failure detectors on wide area network," in *DSN*, 2005.

[50] R. Ceretta Nunes and I. Jansch-Porto, "Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin," in *DSN*, 2004.

[51] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *IEEE Trans. Comput.*, vol. 51, no. 5, 2002.

[52] D. Dzung, R. Guerraoui, D. Kozhaya, and Y.-A. Pignolet, "Never say never - probabilistic and temporal failure detectors," in *IPDPS*, 2016.

APPENDIX A
DETAILS OF VIEWSNOOP'S GUARANTEES

ViewSnoop guarantees property SYMS 1 due to two reasons. First, the *id* of the view to be installed at control cycle r is r, the control cycle number itself. Second, crashed hosts do not recover and if host A excludes host B at cycle r, then A does not install any view at cycles $> r$ where B is considered alive (after excluding B, A ignores any heartbeats from B). SYMS 3 is ensured in *ViewSnoop* of the following reasons:

- A host h that fails at cycle r, stops sending heartbeat message from cycle $r+1$ onward.
- Since h sends no heartbeats at cycles in $[r+1, \infty[$, then h is included in the suspected list of all host alive at cycles in $[r+1, \infty[$.
- As indicated in Section IV-A1, a host h suspected all alive hosts at cycle $r+1$ is declared as failed at cycle $r+2$.

ViewSnoop guarantees property SYMS 5, since: (i) every host broadcasts to all other hosts a heartbeat at every cycle and (ii) a host h_i receiving a heartbeat from a non-excluded host h_j at cycle r declares h_j as alive. In this sense, any host at cycle r has a positive probability (probability of heartbeats being not dropped) of installing a view containing the set of hosts that have not crashed or been excluded up to cycle r.

We detail in what follows how to derive the probabilistic guarantees, p_{agree} and $p_{accurate}$, with which *ViewSnoop* satisfies properties SYMS 2 and SYMS 4, and thus prove Lemma 2 and Lemma 3.

A. Proof of Lemma 2

Consider host A and two sets of hosts:

1) π_A: hosts which do not have A in their $local_{suspect}$ list at the beginning of cycle r.
2) $\pi_{\bar{A}}$: hosts which have A in their $local_{suspect}$ at the beginning of r.

Assume that host A is not included in any of these sets. Let $\mathcal{C}$ be the set of alive hosts in $r-1$ and assume that no host has excluded any other host, then $|\mathcal{C}| = |\pi_A| + |\pi_{\bar{A}}| + 1$. Disagreement in cycle r occurs if any condition below holds:

1) Host A does not receive any message from all hosts in π_A and at least one host in $\pi_A \bigcup \pi_{\bar{A}}$ receives a message from $A \bigcup \pi_A$.
2) At least one host in $\pi_{\bar{A}}$ does not receive any message from all hosts in $A \bigcup \pi_A$ and:
   a) Host A hears from at least one host in π_A OR
   b) At least one host in π_A hears from some other host in $A \bigcup \pi_A$.

Condition (1) happens with probability $Prob(1|\pi_A)$:

$Prob(1|\pi_A) =$
P(Host A does not receive any message from all hosts in π_A)
$\times P$(at least one host in $\pi_A \bigcup \pi_{\bar{A}}$ receives a message from $A \bigcup \pi_A$).

In our computations, we do some approximations for presentation simplicity of closed-form expressions. The value of $Prob(1|\pi_A)$ can be approximated as follows:

$Prob(1|\pi_A) \approx P(\text{Host A does not receive any message from all hosts in } \pi_A)$
$\times P(\text{at least one host in } \pi_{\bar{A}} \text{ receives a message from } A \bigcup \pi_A).$

$P(\text{Host A does not receive any message from all hosts in } \pi_A) = (1-p)^{n_i \times |\pi_A|}.$

$P(\text{at least one host in } \pi_{\bar{A}} \text{ receives a message from } A \bigcup \pi_A) =$
$$\sum_{h=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{h} \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^h \times \left[(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-h-1}.$$

So,

$$Prob(1|\pi_A) \approx (1-p)^{n_i \times |\pi_A|} \times \sum_{h=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{h} \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^h \times \left[(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-h-1}.$$

Condition (2) happens with probability $Prob(2|\pi_A)$:

$Prob(2|\pi_A) =$
$P(\text{At least one host in } \pi_{\bar{A}} \text{ does not receive any message from all hosts in } A \bigcup \pi_A)$
$\times [P(\text{Host A hears from at least one host in } \pi_A)$
$+ P(\text{At least one host in } \pi_A \text{ hears from at least one other host in } A \bigcup \pi_A)].$

We approximate $Prob(2|\pi_A)$ as follows:

$Prob(2|\pi_A)$
$\approx P(\text{At least one host in } \pi_{\bar{A}} \text{ does not receive any message from all hosts in } A \bigcup \pi_A)$
$\times P(\text{Host A hears from at least one host in } \pi_A).$

$P(\text{At least one host in } \pi_{\bar{A}} \text{ does not receive any message from all hosts in } A \bigcup \pi_A) =$
$$\sum_{k=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{k} \left[(1-p)^{n_i(|\pi_A|+1)}\right]^k \times \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-k-1}.$$

$P(\text{Host A hears from at least one host in } \pi_A) = 1-(1-p)^{n_i \times \pi_A}.$

So,

$$Prob(2|\pi_A) \approx [1-(1-p)^{n_i \times \pi_A}] \times \sum_{k=1}^{|\mathcal{C}|-|\pi_A|-1} \binom{|\mathcal{C}|-|\pi_A|-1}{k} \left[(1-p)^{n_i(|\pi_A|+1)}\right]^k \times \left[1-(1-p)^{n_i(|\pi_A|+1)}\right]^{|\mathcal{C}|-|\pi_A|-k-1}.$$

$Prob(1|\pi_A)$ and $Prob(2|\pi_A)$ are probabilities conditioned on the probability, $P(\pi_A)$, of having $|\pi_A|$ hosts that do not have A in their $local_{suspect}$ list at the beginning of cycle r. The probability $P(\pi_A)$ can be expressed as:

$$P(\pi_A) = \sum_{|\mathcal{C}|-|\pi_A|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_A|} (1-p)^{n_i(|\mathcal{C}|-|\pi_A|)} \left[1-(1-p)^{n_i}\right]^{|\pi_A|}.$$

As a result the disagreement probability can be estimated as follows:

$$Prob(disagree_A) \approx [Prob(1|\pi_A) + Prob(2|\pi_A)]\, P(\pi_A).$$

The probability to have a disagreement ($p_{disagree}$) is the probability to disagree about at least one alive host:

$p_{disagree} = \sum_{k=1}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{k} [Prob(disagree_A)]^k [1-Prob(disagree_A)]^{|\mathcal{C}|-k}$,

where: $p_{agree} = 1 - p_{disagree}$.

B. Proof of Lemma 3

Let E be the event that some correct host B is excluded, then $Prob(E) = 1 - p_{accurate}$.

We define the following:

1) π_B: the set of hosts which do not have B in their $local_{suspect}$ list at the beginning of cycle r.
2) $\pi_{\bar{B}}$: the set of hosts which have B in their $local_{suspect}$ list at the beginning of cycle r.
3) The set of all alive hosts in cycle r (assuming no host has excluded any other host), $\mathcal{C} = \pi_B + \pi_{\bar{B}} + \{B\}$.

A correct host B gets declared as crashed and thus is excluded by some host if any of the following happens:

1) B does not hear any message from all hosts in π_B. This event occurs with probability:
$$Prob(1|\pi_B) = (1-p)^{n_i \times \pi_B}..$$
2) At least one host in $\pi_{\bar{B}}$ does not hear any message from all hosts in $B \bigcup \pi_B$. This event occurs with probability:
$$Prob(2|\pi_B) = \sum_{r=1}^{|\mathcal{C}|-|\pi_B|-1} \binom{|\mathcal{C}|-|\pi_B|-1}{r} \left[(1-p)^{(|\pi_B|+1)n_i}\right]^r \times \left[1-(1-p)^{(|\pi_B|+1)n_i}\right]^{|C|-|\pi_B|-r-1}.$$
3) At least one host in π_B does not hear any message from all hosts in $B \bigcup \pi_B$. This event occurs with probability:
$$Prob(3|\pi_B) = \sum_{s=1}^{|\pi_B|} \binom{|\pi_B|}{s} \left[(1-p)^{(|\pi_B|)n_i}\right]^s \times \left[1-(1-p)^{|\pi_B|n_i}\right]^{|\pi_B|-s}.$$

$Prob(E|\pi_B)$, the probability that some correct host B is declared crashed given that π_B hosts heard from B, can be approximated by

$$Prob(1|\pi_B) + Prob(2|\pi_B) + Prob(3|\pi_B).$$

We have:

$$Prob(E \cap \pi_B) = Prob(E|\pi_B) * Prob(\pi_B),$$

where $Prob(\pi_B)$ is the probability that exactly $|\pi_B|$ have host B in $local_{suspect}$ list at the beginning of cycle r. Thus,

$$Prob(E) = \sum_{|\mathcal{C}|-|\pi_B|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_B|} Prob(E \cap \pi_B)$$
$$= \sum_{|\mathcal{C}|-|\pi_B|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_B|} Prob(E|\pi_B) \times Prob(\pi_B)$$
$$= \sum_{|\mathcal{C}|-|\pi_B|=2}^{|\mathcal{C}|} \binom{|\mathcal{C}|}{|\mathcal{C}|-|\pi_B|} (1-p)^{n_i(|\mathcal{C}|-|\pi_B|)} \left[1-(1-p)^{n_i}\right]^{|\pi_B|} \times Prob(E|\pi_B).$$

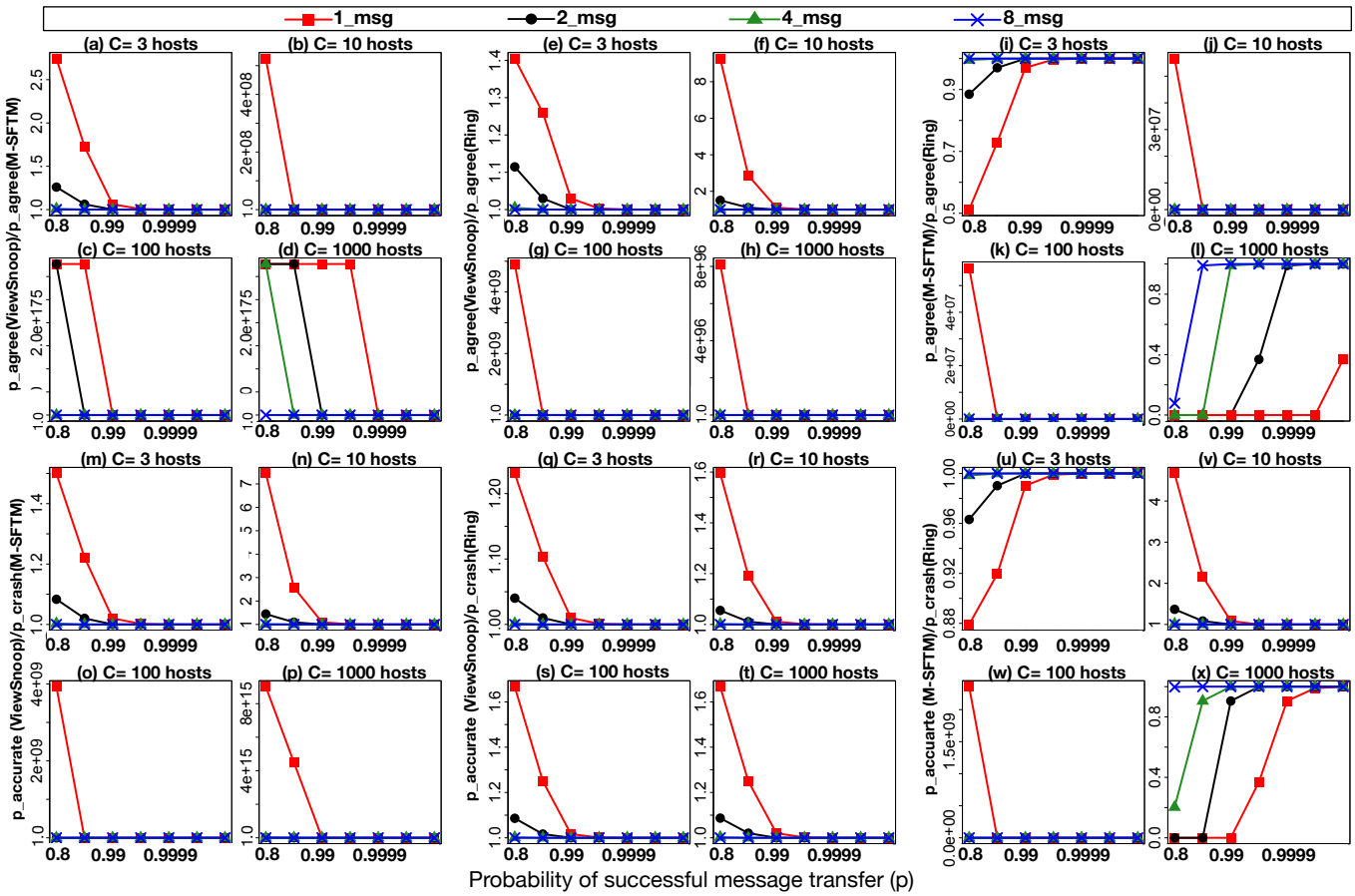

Figure 10. The ratios of the agreement and failure accuracy probabilities of ViewSnoop w.r.t. those of other algorithms.

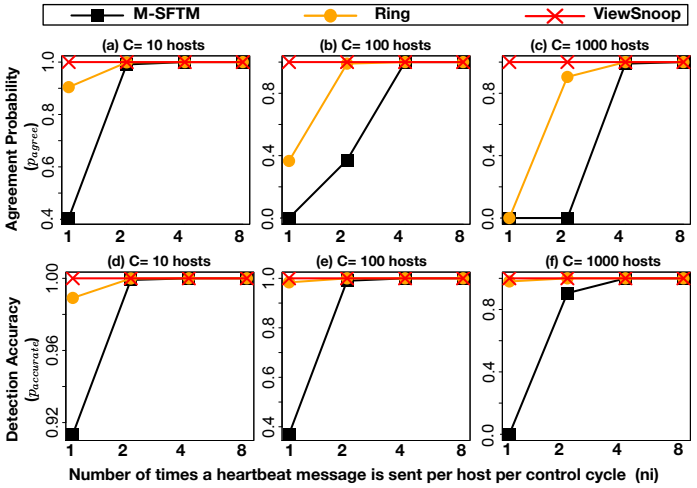

Figure 11. Values of p_{agree} and $p_{accurate}$ of all algorithms versus varying values of n_i for $p = 0.99$.

APPENDIX B
ANALYTICAL RESULTS EXTENDED

In this section, we show more results of our analytic simulation which were omitted in the main part of the paper for space limitations.

We compare the values of p_{agree} and $p_{accurate}$ achieved by *ViewSnoop* versus those obtained using SFTM, M-SFTM and the ring algorithms. Precisely, we simulate the values of p_{agree} and $p_{accurate}$ for all algorithms using $n_i \in \{1, 2, 4, 8\}$ broadcast messages per control cycle. Note that for $n_i = 1$, SFTM and M-SFTM become the same algorithm. Simulations are run for $p \in \{0.8, 0.9, 0.99, 0.999, 0.9999, 0.99999, 0.999999\}$ and $\mathcal{C} \in \{3, 10, 100, 1000\}$, where p is the probability of a host receiving a broadcast message successfully and $\mathcal{C}$ is the number of alive hosts in the system.

In Figure 10(a)-(d) and (e)-(h), we report the ratio of *ViewSnoop*'s agreement probability over that of M-SFTM and that of the ring algorithm respectively. We also report the ratio of *ViewSnoop*'s accuracy probability over the accuracy probability of M-SFTM and of the ring algorithm in Figure 10(m)-(p) and (q)-(t) respectively. The results in these figures are a further support to the conclusion drawn in Section V-A. Namely, *ViewSnoop* offers a more dependable service, compared to SFTM, M-SFTM and the ring algorithm, enhancing thus the availability of a DCS. The significance of this improvement relies on: (i) the number of heartbeats sent every cycle, (ii) the reliability of the communication and (ii) the size of the DCS. For DCSs suffering from communication losses, our algorithm provides superior probabilistic guarantees for critical cyclic control applications (where the number of sent heartbeats per cycle is scarce) compared to mechanisms based on sending simple heartbeats.

In Figure 10(i)-(l) and (u)-(x), we respectively show the

ratio in the agreement probability and the accuracy probability of M-SFTM over the ring algorithm. In Figure 11, we show the agreement probability ((a)-(c)) and the accuracy probability ((d)-(f)) of each of *ViewSnoop*, M-SFTM and the ring algorithm when the probability of successfully receiving a sent message is $p = 0.99$. The results of Figure 11 are reported for $n_i \in \{1, 2, 4, 8\}$ broadcast messages per control cycle and $\mathcal{C} \in \{10, 100, 1000\}$ alive hosts in the system. Note that for $n_i = 1$, SFTM and M-SFTM become the same algorithm. Again these results further validate our conclusions in Section V-B which namely state that:

1) Increasing the number of heartbeats sent per host per cycle of an algorithm implementing SYMS increases the probabilistic guarantees of that algorithm; however this relation does not hold across algorithms; the performance of distinct algorithms cannot be compared solely based on their message complexity.
2) Appending local views to heartbeats allows *ViewSnoop* to increase its probabilistic guarantees and thus the availability of the DCS. The improvement is most significant in large DCSs running critical applications (small n_i).

APPENDIX C
PROOF OF VIEWSOOP'S HIGH PROBABILITY GUARANTEES FOR LARGE SYSTEMS

Lemma 4. *The probabilistic guarantees of ViewSnoop, p_{agree} and $p_{accurate}$, both tend towards 1, as the number of hosts in the system tends towards ∞.*

Proof: Similar to Appendix A, consider a host A and two sets of hosts defined below:

1) π_A: hosts which do not have A in their $local_{suspect}$ list at the beginning of cycle r.
2) $\pi_{\bar{A}}$: hosts which have A in their $local_{suspect}$ at the beginning of r.

Let $\mathcal{C}$ be the set of alive hosts at cycle r. We compute p_{agree} and $p_{accurate}$ when the number of alive hosts is very big, i.e., when $\mathcal{C} \to \infty$.

Hosts in π_A ($\pi_{\bar{A}}$) are those hosts which do not have (have) host A in their $local_{suspect}$ list at the beginning of cycle r. In other words, hosts in π_A ($\pi_{\bar{A}}$) are those hosts which received (did not receive) a heartbeat from host A in cycle $r-1$. Since an infinite number of hosts is alive ($\mathcal{C} \to \infty$), then an infinite number of hosts receive A's heartbeat in cycle $r-1$ and an infinite number of hosts do not receive A's heartbeat in cycle $r-1$, i.e., $|\pi_A|, |\pi_{\bar{A}}| \to \infty$. (An analogy with A sending a message to an infinite number of hosts is flipping a coin an infinite number of times where a head presents a successful message delivery and a tail represents a loss. Flipping a coin an infinite number of times will result in observing an infinite number of heads and an infinite number of tails).

Recall from Appendix A, that disagreement in cycle r occurs if any condition below holds:

1) Host A does not receive any message from all hosts in π_A and at least one host in $\pi_A \bigcup \pi_{\bar{A}}$ receives a message from $A \bigcup \pi_A$.
2) At least one host in $\pi_{\bar{A}}$ does not receive any message from all hosts in $A \bigcup \pi_A$ and:
   a) Host A hears from at least one host in π_A OR
   b) At least one host in π_A hears from some other host in $A \bigcup \pi_A$.

Condition (1) happens with probability $Prob(1|\pi_A)$:

$$Prob(1|\pi_A) = P(\text{Host A does not receive any message from all hosts in } \pi_A) \times P(\text{at least one host in } \pi_A \bigcup \pi_{\bar{A}} \text{ receives a message from } A \bigcup \pi_A).$$

Condition (2) happens with probability $Prob(2|\pi_A)$:

$$Prob(2|\pi_A) = P(\text{At least one host in } \pi_{\bar{A}} \text{ does not receive any message from all hosts in } A \bigcup \pi_A) \times [P(\text{Host A hears from at least one host in } \pi_A) + P(\text{At least one host in } \pi_A \text{ hears from at least one other host in } A \bigcup \pi_A)].$$

We prove in what follows that the probability of any host in $\pi_{\bar{A}} \bigcup A$ not receiving a message from a host in π_A tends to zero, and hence $Prob(1|\pi_A), Prob(2|\pi_A)$ tend to zero as well. The probability that a host in $\pi_{\bar{A}} \bigcup A$ does not hear any heartbeat from all hosts in π_A, given that $|\pi_A|$ hosts do not have A in their $local_{suspect}$ list at the beginning of cycle r, $P(H|\pi_A)$ is: $P(H|\pi_A) = (1-p)^{n_i \times |\pi_A|}$.

However, since $|\pi_A| \to \infty$ then $P(H|\pi_A) \to 0$. This implies that every host in $\pi_{\bar{A}} \bigcup A$ hears a message from some host in π_A and thus $Prob(disagree_A) \to 0$, i.e., p_{agree} is in $1 - \mathcal{O}((1-p)^{|\pi_A|}), \forall\ p \in]0, 1[$.

Following similar reasoning, if host A is correct (does not fail during the entire execution), then the probability that a host H does not declare A as crashed in cycle $r+1$ can be guaranteed if H hears from some host in π_A. This happens with probability: $P_A(H|\pi_A) = 1 - (1-p)^{n_i \times |\pi_A|}$.

Since $\pi_A \to \infty$ when $\mathcal{C} \to \infty$, then $P_A(H|\pi_A) \to 1$ in that case. The probability, $p_{accurate}$, that an excluded host has actually crashed can be restated as the probability of not excluding a correct host. This means $p_{accurate} = P_A(H|\pi_A)$ for some correct host and also tends to 1. ■

These results show that *ViewSnoop* implements the agreement and accuracy probabilities of the SYMS with high probability (tend to 1 as the number of hosts increases).

APPENDIX D
PROBABILISTIC GUARANTEES OF THE RING ALGORITHM

We present in this section the details for approximating the probabilistic guarantees of the ring algorithm presented in Section V. The probability of hosts installing the same view in the ring algorithm boils down to two cases:

1) If every alive host receives at least one heartbeat from the host that precedes it in the ring, i.e., if there are no false suspicions.
2) If all hosts receive the $< id(\text{host}_x), crash >$ of any host, whenever sent.

In the second case, i.e., (2), a host h_i which is falsely suspected agrees to install a view where h_i is not alive in that view. As such, according to the description of the algorithm h_i no longer receives heartbeats from the alive host h_j preceding

it. Since h_i is still alive, despite being excluded form the view, h_i eventually suspects h_j and broadcasts the message $< id(\mathrm{h}_j), crash >$. Upon receiving such messages, we face two cases:

1) Hosts exclude h_j and then h_j leads the host preceding it to get excluded and so on, until all hosts get eventually excluded. This case means that the accuracy of the algorithm drops to zero, as a correct host gets excluded for sure.
2) Hosts ignore messages received from the excluded host h_j. In this case, h_j becomes in disagreement with the rest of hosts in the system.

To this end, the probability of agreement among hosts in the ring algorithm is approximated with the probability that all correct hosts do not get falsely suspected. In other words:

$$p_{agree}(\text{Ring}) = [1 \ - \ (1-p)^{n_i}]^{|\mathcal{C}|} .$$