The Splendors and Miseries of Rounds

Shimi, Adam
adam.shimi@irit.fr

July 7, 2019

1 Introduction

Take a stroll with me through the distributed computing literature: we could scour the proceedings of PODC, DISC, SIROCCO, OPODIS, and more; we could explore the issues of Distributed Computing; or we could skim through the multitude of preprints on arXiv. Yet, I can already predict that with good probability, the paper we choose to read will contain the word "round" or "layer".[1] Is it a lower-bound paper? Then round complexity is probably one measure of performance considered. Is it a combinatorial topology paper? Almost all techniques developped from application of combinatorial topology assume layers of communication, such that an execution can be reduced as successive complexes; here again rounds are essential. Is it a paper on fault-tolerance? Then some broadcasting strategy to ensure decent replication probably uses rounds.

My point is not that all of distributed computing relies on rounds; more that this concept was put to good use in almost all subfields. From this observation, a question naturally follows: why? What make rounds powerful enough to crop up everywhere in the field?

Before we can even hope to answer this question, we must first disentangle rounds from an abusive relationship with another fundamental concept in distributed computing: synchrony. In the beginning, the bond between those two made sense – synchrony allowed an elegant implementation of rounds, and rounds provided much needed structure for synchronous executions. But as the years went by, rounds proved their worth in other contexts. They even collaborated with their first love's opposite: asynchrony. Despite all these accomplishments, synchrony repeatedly refused to let go of its supremacy over rounds. They were made for each other, and the other uses were simply distractions. Rounds were denied value of their own because they were always subservient. And even today, many researchers in the field jump to the synchrony assumption when they read "round".

This need not be. As explained above, rounds are defined and studied in other contexts. Moreover, a concerted effort recently culminated in an abstraction of distributed computing models through rounds transcending most system assumptions. This abstraction only considers which process hears from which other during a given round; this is captured by a predicate representing the model. Then classical models are either proven equivalent in terms of computability to such predicates on rounds, or studied through the predicates on rounds they can implement. In this way, rounds are not only liberated from synchrony's domination, but also guaranteed to never again be tied to such a system-level assumption.

Yet, despite these advances, synchrony still looms over many recent works using rounds. Even papers about rounds as an abstraction of asynchronous models manage to add synchrony to the mix, as we will see in Section 3.

[1] A couple of numbers for 2018: out of 59 articles accepted at PODC, 34 contains "round" or "layer"; 30 out of 49 for DISC; and 11 out of 25 for Distributed Computing.

Therefore, the point of this column is threefold: first, to show the variety in the applications of rounds in distributed computing in Section 2; second, to present the latest work on abstracting distributed models through rounds in Section 3; and third, to propose in Section 4 open directions and problems to improve our understanding of such abstractions.

As the use of the "I" pronoun already tells, this paper does not aim at being an unbiased survey, but instead a position paper about my own perspective on rounds. As such, choices and comments of references reflect their value for making my point, not any judgement about their worth.

2 Traditional Uses of Rounds

2.1 The Essence of Rounds

Rounds provide structure; a structure on which algorithm designers are able to build. This aspect of rounds might be the most obvious, since numerous distributed algorithms leverage this structure, for ease of conception and analysis, as well as for robustness.

Yet what is this structure? Informally, rounds consist in sending messages tagged with the current round number, waiting for some messages with the same number and computing the next state while incrementing the round number. But we are far from a mathematical definition here and the word "message" constrain us to specific forms of communication. An abstraction of this operational definition is necessary for rounds to be objects of study themselves.

Fortunately, this was done more than 35 years ago by Elrad and Francez [19]. They studied the decomposition into layers of concurrent composition of CSP programs. What interested them foremost was a decomposition that was safe, that is, equivalent in terms of behavior with the initial program itself. Just like the parallel composition gives a decomposition in space, they looked for a decomposition in time, a sequence of layers that can be executed one after the other. In order to do so, they defined communication-closed layers: layers such that no communication ever happens from one layer to another. The study of the whole program can then be reduced to the independent study of its communication-closed layers.

Communication-closedness captures the essence of rounds. Operationally, it ensures that processes in round r do not take into account messages from other rounds during this one. If they come from previous rounds, messages will never be used again and are thrown away; if they come from future rounds, messages are buffered until their round arrives. Layers can then be implemented in various ways: round counters and labelled messages in message-passing, snapshot objects in shared-memory,... As long as an algorithm can be decomposed into communication-closed layers, it uses rounds as an underlying structure, whether implicitly or explicitly.

This structure comes in handy for automated verification of distributed algorithms; and this, even in the case of asynchronous rounds. Indeed, Chaouch-Saad et al. [8] prove that many properties one wants to check on distributed algorithms, such as the properties defining consensus, can be proved using communication-closed rounds as if they were synchronous. Such reductions were studied notably by Damien et al. [16] and Gleissenthal et al. [24], in order to verify asynchronous fault tolerant algorithms. Almost all formal methods were applied to verify algorithms with communication-closed rounds: model checking by Chaouch-Saad et al. [8], cutoffs and then model checking by Marić et al. [41], proof assistant (Isabelle/HOL) by Charron-Bost et al. [9], and SMT leveraging inductive invariants by Dragoi et al. [18].

2.2 The Adventures of Rounds

Altough I keep repeating that rounds are not equivalent with synchrony, there is no denying the connection between the two. Almost every synchronous algorithm ever studied use rounds. By synchronous, we actually mean a very specific model: message-passing where processes advance (compute) at the same speed

and there is a known upper bound on communication delay. In such a model, there is only one meaningful way to implement rounds: broadcast, wait for the upper bound on communication delay to receive all messages, then change round.

Rounds are such an important building block for distributed computing that many new models are first defined as synchronous, in order to use the power of the round abstraction. For example, many biological inspired "message-passing" models, such as the beeping model [14], the task-allocation model [13] or the Cellular Bioelectric model [23], are synchronous, in part to use rounds. Many other recent models, such as radio networks [12] or dynamic networks [38], also depend on synchrony for rounds.

On the other hand, rounds also appear in decidedly asynchronous models. Two great examples are fault-tolerant algorithms and algorithms using failure detectors. The former use rounds for state-machine replication, as broadcasting and waiting for a threshold number of messages fits rounds perfectly; see Paxos [40], Zab [34] or Viewstamp Replication [43]. As for the latter, failure detectors are used to decide which processes to wait for. This is why many algorithms using failure detectors rely on rounds, dating back from the original ones by Chandra and Toueg [7] and Chandra et al. [6].

Why is this structure so integral to distributed algorithms? In one sentence, because it makes algorithms analysable sequentially. There is no better example of this power than the combinatorial topology approach to distributed computing [31]. In it, problems and protocols are abstracted into manipulations of high-dimensional objects called simplicial complexes. All possible inputs form a complex, all possible output too, and the problem defines constraints on maps from inputs to outputs. The solvability of the problem reduces to the existence of a map satisfying these constraints.

Rounds appear in the definition of such maps. For many standard models of shared-memory and message-passing, the map from inputs to outputs can always be decomposed into communication-closed layers. Then the existence of the sought-for map depends only on the properties of one layer, and on how they are strung together. This reduction of the combinatorial complexity usually yields a clean and powerful mathematical description, usable for many impossibility results and lower bounds. For example, wait-free shared-memory can be described as the iterated application of an elegant map called a barycentric subdivision [32].

One must be careful, though: even with all the benefits of rounds, not all distributed algorithms use them. Many asynchronous ones only mention point to point messages, usually in order to lower the message count or the waiting time. There are even models of distributed computation totally unsuited to communication-closed layers – population protocols for example. Although rounds are mentionned for complexity issues, this model is fundamentally about one-on-one interactions. The focus of population protocols on minimal memory requirements also makes the cost of implementing round prohibitive.

2.3 In Search of Distributed Time

Another fundamental use of rounds, dating back to the dawn of distributed computing, is the measure of time. Indeed, even the slightest asynchrony makes defining a "step" of a distributed algorithm difficult, as all processes do not work at the same speed. And even if they do, distributed computing is above all else about the communication cost, not the local processing cost.

Counting rounds solves both issues: it defines a step of a distributed algorithm, in a way that captures primarly the communication taking place. Although the exact first use of this idea is difficult to find, we can go back at least to Arjomandi et al. [2] in the early 80's. Since then, it has become a staple of distributed computing research: the majority of lower bounds in the literature involve round complexity.

This abstraction of time is sufficiently deep to have inspired a full-fledged distributed theory of complexity [20], analogous to the sequential one [3, 26]. This theory focuses on various complexity classes for distributed decision problems in Peleg's $\mathscr{LOCAL}$ model [44], a standard synchronous model parameterized by a fixed topology. The concept central to this study is the notion of locality: a decision problem is local iff there exists an algorithm solving it in a constant number of rounds. Those represent truly scalable

solutions, as the number of rounds stay the same when the number of processes increases. Another way to put it is that each process only needs bounded local information to decide.

Just as the class of local decision problems can be seen as an analogous to the class $\mathscr{P}$ of tractable sequential decision problems, many fascinating ideas from computational complexity were adapted to the distributed setting: verification of certificates [20, 36, 27], derandomization [22], hardness of approximation [17],... All these thanks to the underlying rounds providing a robust notion of time.

Of course, like any measure, round complexity can become meaningless in certain settings. In asynchronous models for example, the "real" time taken by each round can vary wildly. The structure of rounds itself – repeated broadcasts – also puts them at odds with other measures of performance such as message complexity. Nevertheless, all the results provided by rounds as a complexity measure are enough to convince anyone that it is a fundamental aspect of distributed computing.

3 Abstracting Distributed Models through Rounds

We saw in the previous section how important rounds are in designing and analysing distributed algorithms. Yet there is an even more important use of rounds that is less known, and the main focus of this paper.

To put it in context, we must first recall one of the peculiarity of distributed computing: the number of non-equivalent models of communication and computation. Every paper in the literature begins with a careful definition of the model to be used. This is because there is no absolute model in distributed computing, no equivalent of Turing machines; change just one detail, and you might get a meaningful model with totally different computability and complexity. Thus distributed computing must deals with its variety of models, one way or another.

But we lack a formal definition of a distributed model. Researchers tend to mix formalism with plain english, making models unusable as mathematical objects. One reason for this is that distributed computing takes more of its inspiration from actual systems than other parts of theoretical computer science like complexity theory. Instead of defining a mathematical class of models to work with, we tend to actually model concrete interesting situations. On the one hand, the field is richer for it; on the other hand, it becomes easy to drown in the sea of models, and forget the relations between them.

Here is where rounds save the day. Because of their clean structure, round-based models are only defined by who might hear from whom at which round. And this is especially easy to formalize mathematically. That is to say, if we limit ourselves to round-based models, the distributed computing landscape changes from a sea at storm to a new continent waiting to be explored and charted.

The only question left is whether round-based models represent a significant fraction of distributed computing models.

3.1 Equivalence with round-based models

Classical message-passing models usually consider faults at the level of components such as processes and links; Santoro and Widmayer [47] were the first to propose a model where only the consequence of faults are described. That is, the model only captures whether a message is received or not, and not why. In addition, we have rounds and synchrony. This gives a simple yet powerful round-based model parameterized by how many and which message losses are allowed at each round.

It turns out that this model is particularly appropriate to capture the computability of classical asynchronous models. First, Raynal and Roy [45] published a short note proving the equivalence of the asynchronous model with at most F permanent crashes with the synchronous round-based model where every process receives at least $N-F$ messages per round, where N is the number of processes. Such an equivalence means that each model can simulate the other, and thus that their power in terms of computability is the same.

This line of research was then pushed further by Afek and Gafni [1]. They generalized the previous model by defining "message adversaries", that is predicates constraining which messages can be lost at which round. With an adversary limited to remove a message from at most one direction for each communication link, they show that the synchronous round-based model is equivalent to the wait-free single-reader multiple-writer shared-memory model. Then Raynal and Stainer [46] extended even further this approach for models using failure detectors, both in message-passing and shared-memory. They provide an adversary equivalent to asynchronous shared-memory augmented with Ω, as well as one for asynchronous message-passing augmented with (Σ, Ω). Moreover, these results outline the relations between various message-adversaries.

We thus know that many fundamental models in distributed computing reduce to round-based ones, at least for questions of computability. Yet I find an issue with the message adversary model: synchrony. More specifically, message adversaries are defined on a synchronous model, but we could replace it with any round-based model satisfying the same properties on receptions at each round. That is to say, synchrony is only an implementation detail here, and thus is superfluous. None of the simulations use the fact that if a message is not received for a round in the synchronous model, it will never be received; they only use the fact that it will never be used in the rest of the algorithm. Such a property can be guaranted even in round-based models on top of asynchronous systems, as it is merely communication-closedness.

Fortunately, there are promising formalisms keeping all the power of message adversaries while abstracting away implementation.

3.2 Abstracting rounds

The first instance of abstracting rounds in the way we mean here is the Round-by-Round Fault Detector model (or RRFD) of Gafni [21]. Executions in this model proceed through rounds implemented by waiting for the messages of processes in a set given by an oracle, the RRFD. Different constraints on communication are then captured by predicates on the output sequences of RRFDs. Among others, Gafni found predicates for asynchronous message-passing models with crashes, wait-free shared-memory and partially synchronous message-passing models. It was extended in the GIRAF model of Keidar and Shraer [35] to capture even more models.

But these two abstractions suffer from a similar issue than the message adversary models. That is, the RRFD is still too operational, as it describes how rounds are implemented, not just their properties. Moreover, the use of fault detectors implicitly declare some components to be the cause of failure. And this does not sit well with clear mathematical formalism for round-based model. As mentionned before, pinpointing the cause of failure forces us to rely on actual crashes and failing links, and thus to consider many uncomparable parameters. Last but not least, RRFDs do not have to be correct. It makes sense in the study of indulgent algorithms [28]; but as general models of computation, this allows weird behaviors such as rounds that never finish.

A solution to these problems is the removal of almost every operational detail: we only consider the messages received at each round. This yields the Heard-Of model of Charron-Bost and Schiper [11], where models are captured by predicates on the sequences of messages actually received. Such a sequence is called a Heard-Of collection, and specify the messages from each round that were received before the end of the round. Because the Heard-Of predicate describes what already happened, there is no risk of waiting forever, and most implementation details of rounds are hidden. And since predicate are completely formal, comparison between model is meaningful.

Such a comparison relation was even defined in the original paper [11], under the name of translation. One predicate A can be translated into another B if there is an algorithm using A that rebroadcast each message a certain number of rounds, and then choose a subset of the received message for the process, such that these sets satisfy predicate B. Basically, we simulate B on top of A by taking multiple rounds of A to gather enough messages and then choosing the ones to keep according to B. It has yet to be studied in depth,

although some translations are given in the original paper [11] and in some papers about specific problems, such as the study of approximate consensus by Charron-Bost et al. [10]. There has also been a study of the limitations of this reduction by Schmid et al. [48].

3.3 Implementing rounds

Which predicates on rounds can be implemented on top of a given distributed model? The various equivalent results mentionned above provide answers for specific cases (when removing synchrony), but not a general approach. And since we want to capture as much distributed models as possible through the Heard-Of model, this is an important question.

Unfortunately, not much has been done in this direction. The only work tackling this issue, to the best of my knowledge, is a paper by my collaborators and I [49]. In it, we focus on the impact of asynchrony for the implementation of rounds. Indeed, in a synchronous system, there is only one meaningful way to implement rounds: wait for the upper bound on communication delay. And this is completely independent of faults, failures, message losses and other parameters. But for an asynchronous model, this bound does not exist, and the waiting condition (called strategy from now on) must depend explicitly on the model.

The focus on asynchrony is done by introducing another kind of sequences of communication graphs: Delivered collections. The difference with Heard-Of collections is that Heard-Of collections capture the messages from a round received before the end of the round of the receiver, while Delivered collections capture which message will be **eventually** delivered, not necessarily in the right round. Delivered predicates, that is predicates on Delivered collections, then describe the possible patterns of deliveries when implementing rounds on top of the original model. Adding asynchrony then reduces to asking which Heard-Of predicates can be implemented on top of the Delivered predicate capturing the model.

Such an implementation follows from a specific strategy, a waiting condition that says when to change round. As long as a strategy never blocks processes forever, it will generate a Heard-Of predicate from a Delivered predicate. The optimal, or dominant, strategy is then the one constraining communication as much as possible, that is the one whose generated Heard-Of predicate is the smallest. Intuitively, adding asynchrony will create more communication behavior than the Delivered predicate allows, since we might not wait for messages that will eventually be delivered. The goal is therefore to find the smallest overapproximation of the initial Delivered predicate, that is the smallest Heard-Of predicate that can be implemented on it.

In the paper, we then go on to study various Delivered predicates capturing classical models, as well as results for all Delivered predicates concerning classes of strategies. Yet we barely scratched the surface: there is no result about the existence or not of a characterizing Heard-Of predicate for each Delivered predicate, and this notion of implementation does not take into account the possibility for processes to exchange information about the messages they received.

4 Where to go next

As I hope I have shown, there is a whole research area waiting for more input. Rounds as abstractions of distributed computing models are promising, and even if they do not make the cut in the end, their study is bound to yield fundamental insights about distributed computation.

The following is a proposal for three distinct research directions, capturing three angles on the study of round-based models. They are all then expanded into more concrete research problems.

4.1 Making Rounds

Open Research Direction 1 (Implementation of Rounds)**.**
Studying further the links between classical distributed models and the Heard-Of predicates they implement.

Not surprising given how much previous work focus on this question (or a variant): all message adversary papers [45, 1, 46], as well as the paper by my collaborators and I [49].

Nonetheless, there is still a lot of work to do. The first problem is also the most straightforward, as it is taken directly from our paper [49] perspectives: the existence, or not, of a dominant strategy, and thus a characterizing predicate, for every Delivered predicate. As of now, it makes sense to conjecture it is the case, since a counter-example would imply an infinite descent of Heard-Of predicates.

Yet without a proof, there is no result. And even with a counter-example, there might be a characterization large enough to capture most reasonable models.

Open Research Problems 1.1 (Dominant strategy)**.**
Is there a dominant strategy (and thus a smallest Heard-Of predicate) for every Delivered predicate?
If not, find a characterization of the Delivered predicates with one.

But even if such a strategy exists in most cases, it will still be hard to find. Instead of doing it by hand, we might look for automated techniques from formal methods. Of course, there is very little hope to solve all cases, but once again a clean set of Delivered predicates and strategy could admit a nice algorithm.

Open Research Problems 1.2 (Automated synthesis and verification of strategies)**.**
Find an algorithm synthetizing a dominant strategy for any Delivered predicate with such a strategy.
If it is undecidable, then characterize a decidable fragment.

The next problem concerns the various computability equivalences obtained between classical models and Heard-Of predicates (or message adversaries). They encompass many models, but hardly all of them. The natural extension would be to show a general computability equivalence between all classical models and their characterizing Heard-Of predicate; but the lack of formalism for the former models is forbiding such a sweeping result. Moreover, rounds are a pretty specific structure, and it would be surprising if no computability power is lost by using them.

Thus another angle of attack is to look for a problem and a model such that the problem is solvable in the model but not in any Heard-Of predicate implementable by the model.

Open Research Problems 1.3 (Computability equivalence)**.**
Is there a problem and a model such that the problem is solvable in the model but not in any Heard-Of predicate implementable in the model?

Turning to complexity, I already explained that round serves to measure time quite robustly. This comes from the additional structure, and thus can incur some costs.

One such cost is in terms of message complexity: the repeated broadcasts are about as inefficient as can be. It is thus reasonable to assume the message complexity increase when going from a classical model to its characterizing predicate for some problems. Similarly, the bit complexity – the quantity of information exchanged – might be higher than in point-to-point algorithms.

Open Research Problems 1.4 (Complexity tradeoffs)**.**
What are the costs of using rounds for solving given problems in terms of message complexity or bit complexity?

Last but not least, the field is missing interesting examples of classical models and their Heard-Of predicates. It is a shame to be limited in this regard, given the enormous amount of variants in distributed computing. The formalism defined by my collaborators and I [49] being tailored for asynchronous message-passing models, going beyond will require new approaches.

Open Research Problems 1.5 (More models under study)**.**
Which Heard-Of predicates capture other models than the classical asynchronous ones ?

- *Shared-memory models*
- *More failure detector models*
- *Partially synchronous models*
- ...

4.2 Extending Rounds

Despite my focus on it, the Heard-Of model might not be the absolute abstraction of round-based models; it remains a great starting point nonetheless. Looking for what can be tweaked is thus an important research direction.

Open Research Direction 2 (Variations on the Heard-Of model)**.**
What can be added/removed to the Heard-Of model to get a more general abstraction?

The most obvious parts of the Heard-Of model that could be relaxed are two assumptions: the set of processes is fixed, and it is known by all. This is because the Heard-Of predicate is known by the processes, and it depends directly on those two pieces of information.

Open Research Problems 2.1 (Dynamicity)**.**
What is a simple and powerful variant of the Heard-Of model capturing dynamicity for processes?

Open Research Problems 2.2 (Knowledge)**.**
What is a simple and powerful variant of the Heard-Of model where processes do not know all other processes?

- *Port-numberings, such as the variants in Hella et al. [30]*
- *Anonymous processes*
- *Different network topologies, where only some neighborhood is known*

The Heard-Of model was first used to study consensus algorithms. Consensus is a distributed task, that is it only happen once. But many interesting problems are more of the repeated kind – implementing distributed objects for example. As is, the Heard-Of model is not able to capture algorithms for these types of problems.

What is missing is the way to get the new inputs/requests.

Open Research Problems 2.3 (Repeated task)**.**
Extend the Heard-Of model to allow the writing of algorithms solving repeated tasks, such as distributed objects.

Another underlying assumption of the Heard-Of model is its deterministic nature. Because algorithms are deterministic, an execution depends only on the algorithm, the initial configuration and the Heard-Of collection. Removing determinism would make executions much harder to describe, and maybe to manipulate.

Yet in some cases, we do not want determinism. Let us say we write an algorithm for a distributed set. Then we might want to give an operation for removing an element of the set, and describe the algorithm without expliciting which one. This is impossible for a deterministic algorithm.

Open Research Problems 2.4 (Non-determinism)**.**
Incorporate non-determinism in the Heard-Of model.

Given a Heard-Of predicate, the actual communication behavior can be any collection satisfying the predicate. This is especially useful for impossibility results and lower bounds, since it means one needs only to focus on the worst cases.

But is this a realistic assumption? Or equivalently, can we incorporate more information into the predicate? For example, we might want to extend the predicate capturing at most F crashes by saying that the probability of a new crash decrease as the negative exponential of the number of crashes at this point. That is, few crashes is way more common than many.

One way to do this is to endow the Heard-Of predicate with a probabilistic distribution. Another requires a generative model giving the distribution of the next communication graph from the prefix. Either way, adding probabilities to the representation of round-based models will permit us to capture more intricacies about distributed computation.

Open Research Problems 2.5 (Probabilistic Heard-Of)**.**
Add probability to Heard-Of predicates.

4.3 Leveraging Rounds

Last but not least, a good abstraction should not only be elegant, but also powerful. It is therefore crucial to find many deep results on the Heard-Of model, of the kind studied in distributed computing: impossibility results, lower bounds and optimal algorithms.

Of course, there are already results: new algorithms for consensus [11], a characterization of approximate consensus solvability [10], a characterization of consensus solvability for memoryless predicates [15], a point-set topology characterization of consensus solvability for general predicates [42], a couple of results on k set-agreement [5, 33, 25] and some dynamic network related results [39, 38, 37, 29], among others. Results on the $\mathscr{LOCAL}$ model are also transposable to the Heard-Of, but they only concern predicates that never change from one round to the other.

Thus a lot of work needs to be done to study further computability and complexity on Heard-Of predicates.

Open Research Direction 3 (General computability and complexity results for HO)**.**
Furthering the understanding of computability and complexity for important problems depending on the Heard-Of predicate.

What lacks from the already proven results is a common approach. Each paper adapts itself to the problem studied and the specific Heard-Of predicates, without links between the proofs.

On the other hand, finding a framework for studying computability and complexity depending on mathematical properties of the predicates would be a huge step forward. There are currently multiple works in submission looking at this question from the angle of point-set topology and combinatorial topology respectively.

Open Research Problems 3.1 (Computability approach)**.**
Find a general approach to prove computability results on Heard-Of predicates.

Open Research Problems 3.2 (Complexity approach)**.**
Find a general approach to prove complexity results on Heard-Of predicates.

The ultimate level of framework for proving a result is automated verification; that is to say, the ability to prove some properties of distributed algorithms in a purely automated way. I mentionned in the first section a few works in this direction, from the model checking of Chaouch-Saad et al. [8] to the cutoffs for consensus of Marić et al. [41]. These all verify the correctness of distributed algorithms.

Here too, having a framework specifically tuned to prove such correctness properties will be a huge boost to the theoretical and practical use of round-based model to think about distributed algorithms.

Open Research Problems 3.3 (Automated verification of correctness for algorithms on Heard-Of predicates)**.**
Build a framework using formal methods to verify the correcteness of algorithms for the Heard-Of model.

Although there is already a lot of work on the Heard-Of model or equivalent ones, all algorithms thus studied are deterministic, to the best of my knowledge. This fits with the intuition behind the Heard-Of model, as an execution is then completely determined by the algorithm, the initial state and the Heard-Of collection.

But randomized algorithms represent a considerable part of distributed computing. Other round-based models, such as the beeping model of Cornejo and Kuhn [14], take advantage of randomness to find very elegant and powerful algorithms. It is also well-known that randomization can change computability dramatically, for example in the case of consensus in asynchronous crash-prone message-passing [4]. Therefore, we should also study its power in the Heard-Of model.

Open Research Problems 3.4 (Randomized algorithms)**.**
Study the computability and complexity of important distributed problems for randomized algorithms in the Heard-Of model.

Finally, we need to be able to compare one Heard-Of predicate with another. This will yield notions of weakest predicate for a given problem, of equivalence between models and of the price of reduction.

This comparison is already defined by Charron-Bost and Schiper [11] in their original paper: translations. But since this paper, almost no work went into the study of translations. There are no building blocks to build translation, no strong results on whether two given predicates can translate into one another and what not. That is to say, translations are defined but not yet understood.

Open Research Problems 3.5 (Reductions)**.**
Study translations between Heard-Of predicates.

- *General results on whether such a translation exists between two predicates*
- *"Archetypal" translations one can use to try and build one*
- *The complexity overhead generated by a translation, and the classes of translations according to this cost.*

5 Conclusion

Rounds are fundamental to distributed computing. They appear in synchronous and asynchronous algorithms, provide a robust measure of time for complexity, and help structure algorithms for both issues of computability and verification.

What is more, rounds are also able to unify many disparate models of distributed computing. Through predicates on the message received at each rounds, models such as the Heard-Of one capture the essence of classical distributed computing models, while allowing deep mathematical study and sweeping results.

This approach is still in its infancy. We still have to discover its true power and limits, to extend it to the fullest and to apply it completely to questions of interest in distributed computing. That is why I believe this line of research deserves more awareness and work from the community.

The reward at the end of the line will surely repay the effort to get there.

References

[1] Yehuda Afek and Eli Gafni. A simple characterization of asynchronous computations. *Theor. Comput. Sci.*, 561:88–95, January 2015. doi:10.1016/j.tcs.2014.07.022.

[2] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. A difference in efficiency between synchronous and asynchronous systems. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 128–132. ACM, 1981. doi:10.1145/800076.802466.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1st edition, 2009.

[4] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 27–30. ACM, 1983. doi:10.1145/800221.806707.

[5] Martin Biely, Peter Robinson, Manfred Schmid, Ulrich Schwarz, and Kyrill Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science*, 726:41–77, 2018. doi:10.1016/j.tcs.2018.02.019.

[6] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996. doi:10.1145/234533.234549.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996. doi:10.1145/226643.226647.

[8] Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In *Proceedings of the 3rd International Workshop on Reachability Problems*, RP '09, pages 93–106. Springer-Verlag, 2009. doi:10.1007/978-3-642-04420-5_10.

[9] Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. Formal verification of consensus algorithms tolerating malicious faults. In *Stabilization, Safety, and Security of Distributed Systems*, pages 120–134. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-24550-3_11.

[10] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *Automata, Languages, and Programming*, pages 528–539, 2015. doi:10.1007/978-3-662-47666-6_42.

[11] Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, April 2009. doi:10.1007/s00446-009-0084-6.

[12] Imrich Chlamtac and Shay Kutten. On broadcasting in radio networks - problem analysis and protocol design. *IEEE Transactions on Communications*, 33(12):1240–1246, December 1985. doi:10.1109/TCOM.1985.1096245.

[13] Alejandro Cornejo, Anna Dornhaus, Nancy Lynch, and Radhika Nagpal. Task allocation in ant colonies. In *Proceedings of the 28th International Conference on Distributed Computing*, DISC'14, pages 46–60. Springer-Verlag, 2014. doi:10.1007/978-3-662-45174-8_4.

[14] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *Proceedings of the 24th International Conference on Distributed Computing*, DISC'10, pages 148–162. Springer-Verlag, 2010. doi:10.1007/978-3-642-15763-9_15.

[15] Étienne Coulouma, Emmanuel Godard, and Joseph Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theoretical Computer Science*, 584:80–90, 2015. Special Issue on Structural Information and Communication Complexity. doi:10.1016/j.tcs.2015.01.024.

[16] Adrien Damien, Cezara Dragoi, Alexandru Militaru, and Josef Widder. Reducing asynchrony to synchronized rounds. *CoRR*, abs/1804.07078, 2018. URL: http://arxiv.org/abs/1804.07078, arXiv:1804.07078.

[17] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 363–372. ACM, 2011. doi:10.1145/1993636.1993686.

[18] Cezara Drăgoi, Thomas A. Henzinger, and Damien Zufferey. Psync: A partially synchronous language for fault-tolerant distributed algorithms. *SIGPLAN Not.*, 51(1):400–415, January 2016. doi:10.1145/2914770.2837650.

[19] Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3):155–173, 1982. doi:10.1016/0167-6423(83)90013-8.

[20] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35:1–35:26, October 2013. doi:10.1145/2499228.

[21] Eli Gafni. Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 143–152. ACM, 1998. doi:10.1145/277697.277724.

[22] Mohsen Ghaffari, David G. Harris, and Fabian Kuhn. On derandomizing local distributed algorithms. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 662–673, October 2018. doi:10.1109/FOCS.2018.00069.

[23] Seth Gilbert, James Maguire, and Calvin Newport. On bioelectric algorithms: A novel application of theoretical computer science to core problems in developmental biology. *CoRR*, abs/1809.10046, 2018. URL: http://arxiv.org/abs/1809.10046.

[24] Klaus V. Gleissenthall, Rami Gökhan Kici, Alexander Bakst, Deian Stefan, and Ranjit Jhala. Pretend synchrony: Synchronous verification of asynchronous distributed programs. *Proc. ACM Program. Lang.*, 3(POPL):59:1–59:30, January 2019. doi:10.1145/3290372.

[25] Emmanuel Godard and Eloi Perdereau. k-set agreement in communication networks with omission faults. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.OPODIS.2016.8.

[26] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 1st edition, 2008.

[27] Mika Göös and Jukka Suomela. Locally checkable proofs. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 159–168. ACM, 2011. doi:10.1145/1993806.1993829.

[28] Rachid Guerraoui. Indulgent algorithms (preliminary version). In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '00, pages 289–297. ACM, 2000. doi:10.1145/343477.343630.

[29] Bernhard Haeupler and Fabian Kuhn. Lower bounds on information dissemination in dynamic networks. In *Distributed Computing*, pages 166–180. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-33651-5_12.

[30] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 185–194. ACM, 2012. doi:10.1145/2332432.2332466.

[31] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann Publishers Inc., 1st edition, 2013.

[32] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, November 1999. doi:10.1145/331524.331529.

[33] Denis Jeanneau, Thibault Rieutord, Luciana Arantes, and Pierre Sens. Solving k-set agreement using failure detectors in unknown dynamic networks. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1484–1499, 2017. doi:10.1109/TPDS.2016.2608829.

[34] Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN '11, pages 245–256. IEEE Computer Society, 2011. doi:10.1109/DSN.2011.5958223.

[35] Idit Keidar and Alexander Shraer. Timeliness, failure-detectors, and consensus performance. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 169–178. ACM, 2006. doi:10.1145/1146381.1146408.

[36] Amos Korman and Shay Kutten. On distributed verification. In *Proceedings of the 8th International Conference on Distributed Computing and Networking*, ICDCN'06, pages 100–114. Springer-Verlag, 2006. doi:10.1007/11947950_12.

[37] Fabian Kuhn, Thomas Locher, and Rotem Oshman. Gradient clock synchronization in dynamic networks. *Theory of Computing Systems*, 49(4):781–816, Nov 2011. doi:10.1007/s00224-011-9348-1.

[38] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 513–522. ACM, 2010. doi:10.1145/1806689.1806760.

[39] Fabian Kuhn, Yoram Moses, and Rotem Oshman. Coordinated consensus in dynamic networks. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 1–10. ACM, 2011. doi:10.1145/1993806.1993808.

[40] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. doi:10.1145/279227.279229.

[41] Ognjen Marić, Christoph Sprenger, and David Basin. Cutoff bounds for consensus algorithms. In *Computer Aided Verification*, pages 217–237. Springer International Publishing, 2017. doi:10.1007/978-3-319-63390-9_12.

[42] Thomas Nowak, Ulrich Schmid, and Kyrill Winkler. Topological characterization of consensus under general message adversaries. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, 2019.

[43] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 8–17. ACM, 1988. doi:10.1145/62546.62549.

[44] David Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[45] Michel Raynal and Matthieu Roy. A note on a simple equivalence between round-based synchronous and asynchronous models. In *Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing*, PRDC '05, pages 387–392. IEEE Computer Society, 2005. doi:10.1109/PRDC.2005.10.

[46] Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 166–175. ACM, 2013. doi:10.1145/2484239.2484249.

[47] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89*, pages 304–313. Springer-Verlag New York, Inc., 1989. doi:10.1007/BFb0028994.

[48] Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. On the strongest message adversary for consensus in directed dynamic networks. In *Structural Information and Communication Complexity*, pages 102–120. Springer International Publishing, 2018. doi:10.1007/978-3-030-01325-7_13.

[49] Adam Shimi, Aurélie Hurault, and Philippe Quéinnec. Characterizing asynchronous message-passing models through rounds. In *22nd International Conference on Principles of Distributed Systems (OPODIS 2018)*, volume 125 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, 2018. doi:10.4230/LIPIcs.OPODIS.2018.18.