The Distributed Computing Column

by

Stefan Schmid

University of Vienna
Währinger Strasse 29, AT - 1090 Vienna, Austria
schmiste@gmail.com

In this issue of the distributed computing column, Kyrill Winkler and Ulrich Schmid present an interesting survey of recent research trends on consensus protocols in dynamic distributed systems. Enjoy the new column and many thanks to the authors for their contribution to the Bulletin!

An Overview of Recent Results for Consensus in Directed Dynamic Networks

Kyrill Winkler[*]
TU Wien, Vienna, Austria
kwinkler@ecs.tuwien.ac.at

Ulrich Schmid
TU Wien, Vienna, Austria
s@ecs.tuwien.ac.at

1 Dynamic Networks Overview

In this article, we provide a brief overview of recent advances on the topic of solving consensus in dynamic networks of distributed computing systems. A more detailed version of our exposition will appear in [33], where details and all of the proofs as well as additional results can be found.

Dynamic networks arise when participants communicate by exchanging messages that may potentially get lost. We tend to think of these participants as small computing devices, which are aptly called "processes", however, this is not the point: The core issues investigated are those of local uncertainty due to incomplete information, which arise in every distributed environment, irrespective of the exact nature of the "processes". A related field is parallel computing, however there the focus is on how to distribute some computing task to multiple computing entities, typically coupled via shared memory, such as multiple cores of a processor, in order to perform the computation faster and more efficient. In contrast, distributed computing typically studies what can still be solved in spite of, and, what is impossible due to, faults occurring in a network-coupled system.

One classic failure model in distributed computing are process crashes, where, at some unknown point in time, a process may simply seize to operate. Crashes are usually permanent and are particularly insidious because of two reasons: First, a process may crash after sending a message only to a subset of the other processes, thereby creating local uncertainty of who still received its last message. Secondly, in an asynchronous system, where there are no bounds on the message delays or the time it may take a process to complete its computation, there is no way to locally distinguish a crashed process from a painfully slow one. In contrast,

[*]Kyrill Winkler has been supported by the Austrian Science Fund (FWF) under project ADynNet (P28182) and RiSE/SHiNE (S11405).

dynamic networks usually consider transient communication faults, where links may recover after being faulty. In this sense, crashes are almost a special case of a dynamic network, where at some point in time a process seizes to send any messages.

Our focal point here is a special notion of dynamic networks, where communication is controlled by an entity, called message adversary. This paradigm conceptually assumes that every single message delivery is under the control of an adversary that tries to foil the effort of the processes to solve a distributed computing task. If the adversary has a winning strategy that renders the task unsolvable, irrespective of the algorithm employed by the processes, we say that the task is impossible under the adversary. If, on the other hand, there exists an algorithmic winning strategy for the processes, we say the problem is solvable under the adversary. The main topic of this article are the circumstances under which a given distributed computing task is solvable under a message adversary and under which it is impossible. Clearly, every truly distributed task is impossible if the message adversary is not restricted in some way, since it may simply suppress all messages otherwise. Thus, we will usually assume an adequately restricted message adversary that, ideally, allows us to precisely pin down the relevant parameters for which solvability and impossibility can be established. A concrete example for this can be seen in Figure 6.

We will restrict our attention to synchronous dynamic networks, where the computation evolves in lock-step synchronous rounds. Each round consists of a phase of communication where every process attempts to broadcast its message to all other processes. It is here that the message adversary determines which messages are delivered and which are lost forever. After the communication phase ends, the processes perform some computation, which may depend on the messages that they just received as well as their current state. The rules of this computation are given as a deterministic algorithm, one for each process, which we will specify in pseudo-code for convenience. The communication in a given round, controlled by the message adversary, can be concisely expressed as a directed graph G_r, called the communication graph, where the set of vertices consists of the processes and an edge from p to q in the graph means that q has received the message from p in round r. As we shall see, an essential quality of a communication graph is whether it is *rooted*, i.e., whether it contains a rooted spanning tree. A given run, dictated by the message adversary, effectively consists of a sequence of communication graphs $(G_r)_{r>0}$, one for each round, called a communication pattern. It thus makes sense to describe a message adversary as a set of communication patterns, called the *admissible communication patterns*. Perhaps the most important insight about running deterministic algorithms in these kinds of systems is that the states of the processes in a given round are completely determined by the initial states of the process and the communication pattern generated by the

$$\begin{tikzcd}
& p_2 \arrow[dr] & & p_4 \arrow[dr] & \\
p_1 \arrow[ur] & & p_3 \arrow[ll] \arrow[ur] & & p_5
\end{tikzcd}$$

G_1 for round 1

$$\begin{tikzcd}
& p_2 \arrow[dr] & & p_4 & \\
p_1 & & p_3 \arrow[ll] \arrow[ur] & & p_5
\end{tikzcd}$$

G_2 for round 2

$$\begin{tikzcd}
& p_2 & & p_4 & \\
p_1 & & p_3 \arrow[ll] \arrow[ul] \arrow[ur] \arrow[rr] & & p_5
\end{tikzcd}$$

G_3 for round 3

Figure 1: Communication graphs G_1, G_2, G_3 for 3 rounds.

$$\begin{tikzcd}
& \mathbf{p_1} \arrow[d] & \\
& \mathbf{p_2} \arrow[dl] \arrow[d] \arrow[dr] & \\
p_3 & p_4 & p_5
\end{tikzcd}$$

Round 1

$$\begin{tikzcd}
& \mathbf{p_1} \arrow[d] & \\
& \mathbf{p_5} \arrow[dl] \arrow[d] \arrow[dr] & \\
p_3 & p_4 & \mathbf{p_2}
\end{tikzcd}$$

Round 2

$$\begin{tikzcd}
& \mathbf{p_1} \arrow[d] & \\
& \mathbf{p_3} \arrow[dl] \arrow[d] \arrow[dr] & \\
\mathbf{p_2} & p_4 & \mathbf{p_5}
\end{tikzcd}$$

Round 3

$$\begin{tikzcd}
& \mathbf{p_1} \arrow[d] & \\
& \mathbf{p_4} \arrow[dl] \arrow[d] \arrow[dr] & \\
\mathbf{p_2} & \mathbf{p_3} & \mathbf{p_5}
\end{tikzcd}$$

Round 4

Figure 2: A communication pattern where it takes $O(n)$ rounds until some node broadcasts, despite a constant hop distance in every single graph (bold nodes represent processes that heard from p_1 after the depicted communication graph was applied).

message adversary.

An example for the first three rounds of a communication pattern can be seen in Figure 1. Even though there is no direct connection from p_1 to p_5 and p_5 is even disconnected most of the time, there is still an information flow from p_1 to p_5 in the following sense: If every process p_i initially holds some piece of information, e.g. an integer x_i, and executes a simple store and forward algorithm, i.e., it stores and forwards all integers received so far in some set S_i, we have $x_1 \in S_5$ at the end of round 3. Clearly, x_1 is forwarded to p_2 in round 1, which in turn forwards it to p_3 in round 2. Finally, p_5 receives x_1 from p_3 in round 3.

It is important to realize that local properties of the communication graphs do not necessarily have direct implications on the communication pattern, even though this might seem plausible at first glance. For example, one may be tempted to conjecture that a maximum radius of 2 in every communication graph guarantees that all processes have heard from some process after a constant number of rounds. This is not the case, however: Consider, for example, the three-round communication pattern for processes $p_1, \ldots, p_5$ shown in Figure 2. Herein, G_1 is a directed tree of height 3, with a single root node p_1 and a single node in the second level. In the following rounds, this second level node switches places with a new node from the third level. In this scenario, p_4 has not heard from p_1 by the end of round 3, even though the length of the path from p_1 to any other process is $\leqslant 2$ in every G_r. It is not hard to generalize this example to a set of n processes

$p_1, \dots, p_n$ to see that it takes actually $O(n)$ rounds until every process heard from p_1 even though every communication graph has a constant radius of 2.

Throughout this article, we focus on the classic distributed consensus problem, where each process is assigned an input value and eventually needs to irrevocably decide on an output value, which must be the same for all processes. It is interesting to note that, under the message adversaries considered here, given an algorithm, the only variance in the initial states of the processes comes from different input value assignments. Therefore, each run of a deterministic consensus algorithm is actually completely determined by this input value assignment, together with the communication pattern.

2 Justification

The ability to achieve agreement in a distributed computing system where communication is highly unreliable is interesting both from a theoretical and a practical perspective. For example, consensus is required for ensuring consistency in replicated databases, which are distributed across a number of servers, or for consistent mode switches e.g. in distributed reconfiguration. The interest in solving consensus in dynamic networks comes from the increasing pervasiveness of wireless and even mobile devices, which are subject to unreliable communication and energy constraints. A rigorous theoretical understanding of what can and cannot be computed in dynamic networks is, to some degree, necessary for their design and optimization.

Mobility Mobile ad-hoc networks [19] consist of highly mobile devices that move in a fashion that is hard to predict. This may incur that two participants that could communicate flawlessly with each other at one point in time due to their close proximity may not be able to directly communicate at all soon after if they have moved apart too far. Dynamic networks excel in expressing this issue, as the respective edge may simply disappear from one round to the next in the communication graph. In fact, the network topology may change completely from round to round.

Directed Communication A common paradigm in computer networks is to assume that links are bidirectional, in the sense that if p hears from q at some point in time, then also q hears from p. While this is certainly justified in wired “landline” networks, where the connection is present in both directions or not at all, ensuring this in a wireless setting comes at a price: localized fading or interference phenomena [27, 17] such as the capture effect or near-far problems [32] lead to communication that is inherently directed. Guaranteeing bidirectional communication here necessitates spending additional resources, which may be undesirable in many cases, especially if solutions can also be achieved with unidirectional forms of communication. Furthermore, there are instances such as disaster relief applications [22] where strong communication guarantees may simply not be achievable at all.

Message Adversaries A message adversary that can adapt to varying operating conditions and even to the behavior of some given algorithm is a very strong assumption, which offers several advantages: First and foremost, it allows to model systems that suffer from uncoordinated boot-up sequences or systems that must recover from massive transient faults: Wireless network connectivity can be expected to improve over time here, e.g., due to improving clock synchronization quality. Assuming a message adversary that behaves differently during stable and unstable periods is obviously advantageous in terms of coverage. A solution algorithm that can cope with such an adaptive adversary succeeds in every run, which makes the solution also suitable for safety-critical applications where failures would be catastrophic. Furthermore, such solutions also work if message delivery is vulnerable from a security point of view, i.e., if message loss may, at least to some extent, be in fact caused by a malicious entity that tries to attack the system.

Synchronous Execution Thanks to modern communication technology [31], assuming the ability to implement lock-step synchronous execution of all processes is not unreasonable nowadays. As synchronized clocks are typically required for basic communication in wireless systems anyway, e.g., for transmission scheduling and sender/receiver synchronization, global synchrony is reasonably easy to achieve: It can be integrated directly at low system levels as in 802.11 MAC+PHY [1], provided by GPS receivers, or implemented by means of network time synchronization protocols like IEEE 1588 or FTSP [23].

We hope that the above justifications for message adversaries have sufficiently piqued the curiosity of the reader to dive into the results presented in the following sections. The algorithms given may stimulate the development of more practical solutions for a real system that can be approximated with satisfactory accuracy by a given message adversary, and the characterization theorems allow answering the question whether distributed consensus can be solved in a dynamic network with certain communication guarantees at all.

3 Related Work

Consensus characterizations for distributed computing systems have existed for a long time, at least since the highly influential result [15] that showed the impossibility of consensus in an asynchronous system even if just a single crash failure may occur. In more detail, it has been shown that every potential solution algorithm has an admissible, forever undecided run. This proof is of particular technical interest, as it introduced the now-standard bivalence proof technique, which essentially shows that there is a bivalent initial configuration and each bivalent configuration has a bivalent successor configuration. We will use this formalism heavily for our own impossibility proofs, albeit in a less subtle fashion. In addition to this, it has been shown in [15] that a majority of correct processes suffices to solve consensus in an asynchronous system if processes are either correct or initially dead. The proposed solution algorithm employs a communication graph approximation, in which an initial clique that has no incoming edges from outside of the clique can be found. Much of our employed methodology is closely related to this idea and may even be seen as a translation of this principle to the message adversary model.

The impossibility of asynchronous consensus with a single faulty process from [15] has been generalized in several directions. In [9], the technique from [15] was translated from consensus to decision tasks in general, for which a precise characterization was given. This characterization rests on the notion of input, output and decision vectors (where the i-th component represents the input, output and decision of p_i) and their graph. The core result from [9] is that a decision task is solvable despite a single crash failure if, and only if, the graph of the decision vectors is connected and, for every partial input vector, a covering decision vector exists and can be computed.

Studying consensus in synchronous message passing systems subject to link failures dates back at least to the seminal paper [26] by Santoro and Widmayer; generalizations have been provided in [29, 12, 8, 13, 11] (see below for more details). In all these papers, consensus, resp. variants thereof, are solved in systems where, in each round, a digraph is picked from a set of possible communication graphs. The term message adversary was coined by Afek and Gafni in [2] for this abstraction.

A different approach for modeling dynamic networks has been proposed in [20]: T-interval connectivity guarantees a common subgraph in the communication graphs of every T consecutive rounds. [21] studies agreement problems in this setting. Note that solving consensus is relatively easy here, since the model assumes bidirectional and always connected communication graphs. In particular, 1-interval-connectivity, the weakest form of T-interval connectivity, implies that all nodes are able to reach all the other nodes in the system.

There have also been developed several “round-by-round” frameworks, which allow to relate models of computation with different degrees of synchrony and failures. Examples are round-by-round fault detectors by Gafni [16], the GIRAF framework by Keidar and Shraer [18], the HO model by Charron-Bost and Schiper [12], and the hybrid failure model by Biely et. .al. [8].

Oblivious message adversaries

Perhaps one of the earliest characterizations of consensus solvability in distributed computing systems prone to communication errors is [26], where it is shown that consensus is impossible if up to $n - 1$ messages may be lost each round. The reason for this was identified to be that communication patterns in this case are *adjacency-preserving* and *continuous*, the combination of which allows an inductive construction of a perpetually bivalent execution, i.e., an execution where consensus can never be solved, independent of the solution algorithm employed. This proof is in the spirit of the classic consensus impossibility proof from [15], even though in the latter, the construction of a forever bivalent execution that is also admissible is harder and necessitates somewhat even more subtle arguments.

The term “oblivious message adversary” for a message adversary that can choose the communication graphs in every round from a fixed set of graphs was coined in [13]. In this terminology, the adversary from [26] may, each round, pick any graph from the set that contains all communication graphs where $n - 1$ or fewer edges are missing. In [13], a property of an equivalence relation on the sets of communication graphs was identified, which captures exactly the source of consensus impossibility in the oblivious setting. It was shown how this property can be used to find a necessary and sufficient condition for solving consensus. While this result was certainly a major inspiration for our work, to the best of our knowledge there is no way to generalize it directly to non-oblivious message adversaries. Nevertheless, it provides an exact characterization for consensus solvability under oblivious message adversaries, thereby answering this question and providing a significant generalization of the classic result from [26]. As stated above, the latter can be seen as a special case of [13], even though we should note that the agreement problem considered in [26] was more general than consensus.

Non-oblivious message adversaries

In the world of oblivious message adversaries, as well as in the original notion of message adversaries from [2], there is no notion of eventually stabilizing behavior of dynamic networks. One instance where such stabilizing behavior is described, is the message adversary that guarantees eventually stable root components, considered in [4]: It assumed communication graphs with a non-empty set of sources and long-living periods of stability $x = 4D + 1$. [5] studies consensus under a message adversary with comparably long-lived stability, which gracefully degrades to general k-set agreement in case of unfavorable conditions. However, this message adversary must also guarantee a certain influence relation between subsequently existing partitions. [30] established a characterization of uniform consensus solvability/impossibility for longer stability periods. In particular, it provides a consensus algorithm that works for stability periods of at least $2D + 1$ but does not require graph sequences where all graphs are rooted. We will present some of these ideas in more detail in Section 8.

Note that the experimental evaluation of a wireless sensor network described in [24] reveals that this assumption holds true, for a properly chosen value of D (in particular, $D = 4$), with a coverage close to 100% both in an indoor and outdoor environment. Whereas one cannot obviously generalize from a single, non-systematic experimental evaluation, these findings nevertheless suggest that the basic assumption of an eventually vertex-stable root component is not unreasonable in practice. [25] used message adversaries that allow a notion of "eventually forever" to establish a relation to failure detectors.

For the special case of $n = 2$, [14] provides a complete characterization of consensus solvability for message adversaries that are not oblivious: Using a bivalence argument, it is shown that certain graph sequences (a "fair sequence" or a special pair of "unfair sequences") must be inadmissible to render consensus solvable, and provided a universal algorithm for this case. However, to the best of our knowledge, a complete characterization of consensus solvability for arbitrary system sizes and general message adversaries did not exist.

4 Model

We consider an ensemble of deterministic state machines, called *processes*, which communicate via message passing over unreliable point-to-point links. Processes have unique identifiers and are typically denoted by p, q, p', q', etc. The operation proceeds in lock-step synchronous rounds $r = 1, 2, 3, \dots$ consisting of a phase of message exchange between the processes, which is followed by a phase of local computations. Similar to, e.g., [21], we use the convention that all operations of round r take place strictly within time $r - 1$ and time r, which results in well-defined and stable states of all processes between the rounds: The *state* of a process at time r is its initial state (specifying the starting values for each variable) for $r = 0$, respectively the state at the end of its round-r computation (describing the content of all variables as well as the messages to be sent) for $r > 0$. The collection of the states of all processes at time r is called the *configuration* C^r, with C^0 denoting the initial configuration.

A *dynamic graph* is a mapping of each round r to a directed graph[1] $G_r = \langle \Pi, E^r \rangle$, called the round-$r$ communication graph. Each node of Π represents a process, and an edge (p, q) in G_r represents that the round-r message of p sent to q is received by q in the same round. Since every process p always successfully receives from itself, all graphs G_r are reflexive, i.e., they contain an edge (p, p) for every process $p \in \Pi$. The *in-neighborhood* of p in G_r, $\mathrm{In}_p(G_r) = \{q \mid (q, p) \in G_r)$ hence represents the processes from which p may have received a message in round r. We stress that the vertex set Π of a given dynamic graph is fixed (but usually not known to the processes) and only the set of edges may vary from round to round and assume that every $p \in \Pi$ has a unique identifier from the set $[1, |\Pi|]$. We often identify a dynamic graph with an infinite sequence σ of consecutive communication graphs and denote its vertex set by Π_σ. When describing a continuous subsequence σ' of σ, ranging from round a to round b, we denote this as $\sigma' = (G_r)_{r=a}^{b}$, where $|\sigma'| = b - a + 1$, with $b = \infty$ for infinite subsequences.

A *message adversary* MA that may suppress certain messages in an attempt to foil the collaboration of the processes is at the core of our model. Formally, it is a set of dynamic graphs, or, equivalently, communication graph sequences, which are called *admissible*. Sometimes it will be convenient to denote explicitly restrictions on the size of the vertex set of the dynamic graphs of a message adversary as the first index of MA. For example, MA_n states that every dynamic graph of MA_n has a vertex set of size exactly n, while $\mathsf{MA}_{\leqslant n}$ denotes that this size is at most n. Conceptually,[2] we assume that processes know *a priori* the specification of the message adversary, hence an algorithm that succeeds under MA must be able to cope with the size restrictions of MA. Since a message adversary is a set of dynamic graphs, we can compare different message adversaries via set inclusion.

We consider the *consensus problem*, where each process p starts with input value $x_p \in \mathbb{N}$ and has a dedicated write-once output variable y_p, where $y_p = \perp$ initially; eventually, every process needs to irrevocably decide, i.e., assign a value to y_p (*termination*) that is the same at every process (*agreement*) and was the input of a process (*validity*). The assignment of the input values for each process is specified in the initial configuration C^0. Given a message adversary MA, a deterministic consensus algorithm $\mathcal{A}$ and a $\sigma \in \mathsf{MA}$, an *admissible execution* or *run* $\varepsilon = \langle C^0, \sigma \rangle$ is a sequence of configurations $C^0, C^1 \ldots$ where for $r > 0$, C^r is the result of exchanging the messages to be sent according to C^{r-1} and G_r, and applying the resulting state transitions specified by $\mathcal{A}$. Since $\mathcal{A}$ is deterministic, the execution ε is uniquely determined by an admissible graph sequence $\sigma \in \mathsf{MA}$ and a corresponding initial configuration C^0. Algorithm $\mathcal{A}$ solves consensus

[1]We sometimes write $p \in G_r$ instead of $p \in \Pi$ to stress that p is a vertex of G_r, and sloppily write $(p, q) \in G_r$ instead of $(p, q) \in E^r$.

[2]As we will see in Section 7 de facto the processes require only knowledge of some key parameters.

under message adversary MA if, for every $\sigma \in$ MA and every input assignment C^0, validity, agreement and termination are all satisfied in the execution $\langle C^0, \sigma\rangle$ of $\mathcal{A}$. We will see that in some cases, the size of the set of processes Π may be different in selected dynamic graphs of MA and the processes must cope with this and the fact that they cannot reliably compute the size of Π. We call a consensus algorithm *uniform* (c.f. [3]) for MA if it solves consensus under MA and MA consists of dynamic graphs of arbitrary size.

As usual, we write $\varepsilon \sim_p \varepsilon'$ if the finite or infinite executions ε and ε' are *indistinguishable* to p (i.e., the state of p at time r is the same in both executions) until p decides. When establishing our lower bounds, we will often exploit that, as outlined above, the configuration at time r is uniquely determined by the initial configuration C^0 and the sequence of communication graphs until round r.

Dynamic graph concepts

First, we introduce the pivotal notion of a *root component* R, often called root for brevity, which denotes the vertex set of a strongly connected component of a graph where there is no edge from a process outside of R to a process in R.

Definition 4.1 (Root Component). *$R \neq \emptyset$ is a root (component) of graph G, if it is the set of vertices of a strongly connected component $\mathcal{R}$ of G and $\forall p \in G, q \in R$: $(p \to q) \in G \Rightarrow p \in R$.*

It is easy to see that every graph has at least one root component. A graph G that has a *single* root component is called *rooted*; its root component is denoted by Root(G). Clearly, a graph G is rooted if and only if contracting its strongly connected components to single vertices yields a rooted tree. Hence, G is weakly connected and contains a directed path from every node of Root(G) to every other node of G.

Conceptually, root components have already been employed for solving consensus a long time ago: The asynchronous consensus algorithm for initially dead processes introduced in the classic FLP paper [15] relies on a suitably constructed initial clique, which is just a special case of a root component.

In order to model stability, we rely on root components that are present in every member of a (sub)sequence of communication graphs. We call such a root component the *stable root component* of a sequence and stress that, although the set of processes remains the same, the interconnection topology between the processes of the root component as well as the connection to the processes outside may vary arbitrarily from round to round.

Definition 4.2 (Stable Root Component). *A non-empty sequence $(G_r)_{r\in I}$ of graphs is said to have a stable root component R, if and only if each G_r of the sequence* *is rooted and* $\forall i, j \in I : \mathrm{Root}(G_i) = \mathrm{Root}(G_j) = R$*. We call such a sequence an* R*-rooted sequence.*

We would like to clarify that while "rooted" describes a graph property, "R-rooted" describes a property of a sequence of graphs.

Given two graphs $G = \langle V, E\rangle$, $G' = \langle V, E'\rangle$ with the same vertex set V, let the *compound graph* $G \circ G' := \langle V, E''\rangle$ where $(p, q) \in E''$ if and only if there exists a $p' \in V$ such that $(p, p') \in E$ and $(p', q) \in E'$.

In order to model information propagation in the network, we use a notion of *causal past*: Intuitively, a process q is in p's causal past, denoted $q \in \mathrm{CP}_p^r(r')$ if, at time r, p holds information (sent either directly or transitively, via intermediate messages) that q sent after time r'. This is closely related to various concepts that have been introduced in the literature (cf. for example [10] and the references therein), such as the heard-of sets from [12] or the temporal distance from [34].

Definition 4.3 (Causal past)**.** *Given a sequence* σ *of communication graphs that contains rounds* a *and* b*, the causal past of process* p *from time* b *down to time* a *is* $\mathrm{CP}_p^b(a) = \emptyset$ *if* $a \geqslant b$ *and* $\mathrm{CP}_p^b(a) = \mathrm{In}_p(G_{a+1} \circ \cdots \circ G_b)$ *if* $a < b$*.*

A useful fact about the causal past is that in full-information protocols, where processes exchange their entire state history in every round, we have $q \in \mathrm{CP}_p^r(s)$ if and only if, at time r (and hence thereafter), p knows already the state of $q \neq p$ at time s.

5 Message Adversaries

First, we introduce the adversary that adheres to *dynamic network depth* D, which gives a bound on the duration of the information propagation from a stable root component to the entire network. We showed in [6, Cor. 1] that always $D \leqslant n - 1$; *a priori* restricting $D < n - 1$ also allows modelling dynamic networks where information propagation is guaranteed to be faster than in the worst case (as in expander graphs [6], for example).

Definition 5.1. $\mathsf{DEPTH}_n(D)$ *is the set of all infinite communication patterns* σ *s.t.* $|\Pi_\sigma| = n$ *and, for every ordered set of* D *rounds* $T = \{r_1, \dots, r_D\}$*, if* $\exists R \subseteq \Pi_\sigma, \forall r \in T$*: the graph* G_r *of* σ *satisfies* $\mathrm{Root}(G_r) = R$*, then* $R \subseteq \mathrm{CP}_p^{r_D}(r_1 - 1)$ *for all* $p \in \Pi_\sigma$*.*

The following liveness property, *eventual stability*, ensures that eventually every graph sequence σ has an R-rooted subsequence $\sigma' \subseteq \sigma$ of length x. This implies that all sequences have a vertex-stable root component that consists of the same set of processes with possibly varying interconnection topology for x consecutive rounds.

Definition 5.2. $\lozenge\mathsf{GOOD}_n(x)$ *is the set of all infinite communication patterns* σ *such that* $|\Pi_\sigma| = n$ *and there exists a set* $R \subseteq \Pi_\sigma$ *and an* R*-rooted* $\sigma' \subseteq \sigma$ *with* $|\sigma'| \geqslant x$*.*

For finite x, $\lozenge\mathsf{GOOD}_n(x)$ alone is insufficient for solving consensus: Arbitrarily long sequences of graphs that are not rooted before the stability phase occurs can fool all consensus algorithms to make wrong decisions. For this reason, we introduce a safety property in the form of the message adversary that generates only rooted graphs. As mentioned above, this implies that every communication graph is weakly connected and there is a single root component, i.e. a non-empty set of nodes from which all nodes are reachable.

Definition 5.3. ROOTED_n *is the set of all infinite sequences* σ *of* rooted *communication graphs such that* $|\Pi_\sigma| = n$*.*

The short-lived eventually stabilizing message adversary $\lozenge\mathsf{STABLE}_{n,D}(D+1)$ used throughout the main part of our paper adheres to the dynamic network depth D, guarantees that every G_r is rooted and that every sequence has a subsequence of at least $x = D+1$ consecutive communication graphs with a stable root component. Since processes are aware under which message adversary they are executing, they have common *a priori* knowledge of the dynamic network depth D and the duration of the stability phase x. Moreover, depending on the variant actually used, they have some knowledge regarding the system size n.

Definition 5.4. *We call* $\lozenge\mathsf{STABLE}_{n,D}(x) = \mathsf{ROOTED}_n \cap \lozenge\mathsf{GOOD}_n(x) \cap \mathsf{DEPTH}_n(D)$ *the eventually stabilizing message adversary with stability period x. For a fixed D, we consider the following generalizations:*

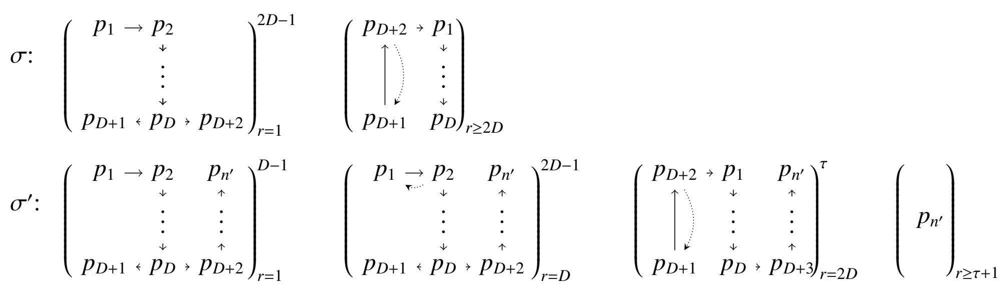
- $\lozenge\mathsf{STABLE}_{<\infty,D}(x) = \bigcup_{n\in\mathbb{N}\setminus\{0,1\}} \lozenge\mathsf{STABLE}_{n,D}(x)$
- $\lozenge\mathsf{STABLE}_{\leqslant N,D}(x) = \bigcup_{n=2}^{N} \lozenge\mathsf{STABLE}_{n,D}(x)$

We observe that $\lozenge\mathsf{GOOD}_n(x) \supseteq \lozenge\mathsf{GOOD}_n(D)$ for all $1 \leqslant x \leqslant D$, hence it follows that $\lozenge\mathsf{STABLE}_{n,D}(x) \supseteq \lozenge\mathsf{STABLE}_{n,D}(D)$.

6 Impossibility Results and Lower Bounds

Even though processes know the dynamic network depth D, for very short stability periods, this is not enough for solving consensus. In Theorem 6.1, we prove that consensus is impossible under $\lozenge\mathsf{STABLE}_{<\infty,D}(2D-1)$ (recall that even if the dynamic graph has a finite set of processes Π, this set is not necessarily known to the processes). That is, if processes do not have access to an upper bound N on the number of processes, solving consensus is impossible if the period x of eventual stability is shorter than $2D$: Here, processes can never be quite sure whether a stable root component occurred for at least D rounds, which is critical, however, since only a duration of D or more rounds guarantees information propagation, according to Definition 5.1.

The core argument of the proof is that an arbitrary correct consensus algorithm $\mathcal{A}$ will fail when exposed to the communication graph sequences σ, σ' from Figure 3. Fix the input values of processes $p_1, \dots, p_{D+2}$ to 0 and let all other processes start with input 1. Because $\mathcal{A}$ satisfies termination, process p_{D+1} eventually, by a time τ, has reached a decision in an execution based on σ. Since the situation is indistinguishable for p_{D+1} from the situation where everyone started with 0, it has to decide 0 by validity. Crucially, p_{D+1} cannot distinguish whether the actual communication graph sequence is σ or σ', thus it decides 0 also in the latter. If n' was chosen sufficiently large, however, process $p_{n'}$ never learns of an input value other than 1. A similar argument as above shows that, by validity, $p_{n'}$ hence eventually decides 1 and thus two values were decided under the communication graph sequence σ'. Clearly, $\mathcal{A}$ does not satisfy agreement, a contradiction to the initial supposition that $\mathcal{A}$ is correct, as σ' is an admissible communication graph sequence.



σ:
p_1 p_2 p_{D+1} p_D p_{D+2}
$2D-1$
$r=1$
p_{D+2} p_1 p_{D+1} p_D
$r \geq 2D$
σ':
p_1 p_2 $p_{n'}$ p_{D+1} p_D p_{D+2}
$D-1$
$r=1$
$2D-1$
$r=D$
p_{D+2} p_1 $p_{n'}$ p_{D+1} p_D p_{D+3}
τ
$r=2D$
$p_{n'}$
$r \geq \tau+1$

Figure 3: Communication graph sequences of Theorem 6.1, where $(G)_{r=a}^{b}$ denotes that G is the communication graph from round a until round b. A dotted edge represents an edge which is in G_i if and only if it is not in G_{i-1}. We assume that there are self-loops and there is an edge from every process depicted in the graph to every process not depicted in the graph.

Theorem 6.1. *Under* $\lozenge\mathsf{STABLE}_{<\infty,D}(x)$ *consensus is impossible for* $0 < x < 2D$.

Theorem 6.1 shows that consensus is impossible under $\lozenge\mathsf{STABLE}_{<\infty,D}(D+1)$, since no process has a bound on the system size. In the remaining paper, we thus study the adversary $\lozenge\mathsf{STABLE}_{\leqslant N,D}(D+1)$, for which we show in the next section that consensus can indeed be solved.

As our next result, we present a lower bound for the duration x of the stable period: We prove that even if there is an upper bound N on the number of processes in the current sequence, consensus is impossible under $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x)$ if $x \leqslant D$ (Theorem 6.2). Note that this result improves the lower bound $x \geqslant D-1$ established in [4, 6] and thus reveals that the latter was not tight. We note, however, that the proof of the earlier result is more general in that it proves bivalence when starting from an arbitrary stabilization round r_0; Theorem 6.2 shows this only for $r_0 = 1$, i.e., when the stable period occurs immediately.

For $N = 2$, our result can be derived from [26], where it was shown that consensus is impossible if at most $n - 1$ messages are lost in each round. In our terminology, this means that consensus is impossible under $\Diamond\mathsf{STABLE}_{\leqslant 2,1}(1)$. For general N, D, Theorem 6.2 below shows that a stability period of D or less rounds is insufficient for solving consensus for arbitrary values of N as well. This result is not subsumed by [26], since the adversary is not restricted by the number of link failures but by the structure of the communication graph sequences.[3]

Informally, the reason for the impossibility is that there are executions where, even with a stability phase of D rounds, there is a process that cannot precisely determine the root component of the stability window. This can be seen when considering the graphs G_a, G_b, G_c, G_d from Figure 4. Fix the input values such that $x_{p_1} \neq x_{p_2}$ and, for each graph G_a, G_b, G_c, G_d, consider the four configurations that result when applying the graph repeatedly for D rounds. As these configurations are connected by an indistinguishability relation, and all processes can become the single root component "forever after" (thereby remaining unable to distinguish the four executions), not all these configurations can be univalent; if they were, the configuration resulting from applying G_a for D rounds would have the same valence as the one resulting from applying G_d for D rounds. An inductive argument, similar to the one employed by [26], shows that this bivalence can be preserved forever and hence no correct consensus algorithm exists.

Theorem 6.2. *There is no consensus algorithm for $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x)$ with $1 \leqslant x \leqslant D$, even if the adversary guarantees that the first D rounds are R-rooted.*

7 Solving Consensus with $D + 1$ Rounds of Stability

We now present our consensus algorithm for $\Diamond\mathsf{STABLE}_{\leqslant N,D}(D + 1)$, where a bound $N \geqslant n$ is known *a priori*. The pseudo-code for the main algorithm is presented in Algorithm 2. It relies on a collection of functions given in Algorithm 1.

Since the detailed correctness proof is somewhat tedious, we first give an informal description of the algorithm where we provide references to the lemmas that correspond to our informal description.

[3]As we will see in the next section, consensus is possible under $\Diamond\mathsf{STABLE}_{n,D}(D + 1)$ even though in some cases (e.g. if $D = n - 1$ and all communication graphs are chains) up to $(n - 1)^2$ messages may be lost in each round!

$$\begin{tikzcd}
p_1 \arrow[d] & p_1 \arrow[d, leftrightarrow] & p_2 \arrow[d, leftrightarrow] & p_2 \arrow[d] \\
p_2 \arrow[d] & p_2 \arrow[d] & p_1 \arrow[d] & p_1 \arrow[d] \\
\vdots \arrow[d] & \vdots \arrow[d] & \vdots \arrow[d] & \vdots \arrow[d] \\
p_{D+1} & p_{D+1} & p_{D+1} & p_{D+1} \\
\mathbf{G_a} & \mathbf{G_b} & \mathbf{G_c} & \mathbf{G_d}
\end{tikzcd}$$

Figure 4: Communication graphs for Theorem 6.2. We assume there is an edge from every process depicted in the graph to every process not depicted in the graph.

Overview. In essence, each process p that executes Algorithm 2 tries to solve consensus by approximating the current graph sequence and keeping track of the relevant partial states of the other processes. Most prominently, this includes their proposal value x, basically their current decision value estimate. If a process observes that in the current graph sequence, the stability phase could have occurred, i.e., it finds a root component R that might have been stable for $D + 1$ rounds, it locks on to the maximum over the proposal values of the members of R. Subsequently, p waits if there is evidence refuting its observation. As soon as p finds such contradictory evidence, it clears its locked-on state. If, on the other hand, p does not find such evidence for a sufficiently long time, it decides on its proposal value. In order for the latter to be a safe decision, the algorithm has a mechanism that guarantees the following: As soon as a process q detects that there might have been a process p that is convinced it holds the correct proposal value, q adopts the proposal value of p. Crucially, q does not enter a locked-on state in this case. The main difficulties of this approach, apart from implementing the usual graph approximation and book-keeping mechanisms, are to ensure that (1) it cannot happen that distinct processes are simultaneously convinced of two different proposal values for too long, that (2) the significant information propagation delays, inherent in this setting, still guarantee a timely adaptation of the proposal of a convinced process in the entire system, while maintaining that (3) the stability period will eventually lead to the same decision at every process.

Detailed description. The essential data structures that each process p maintains are the set of the (reduced) *past process states* S, the *communication graphs approximation* A of the communication graphs that occurred so far, and the *set of participating processes* P of whose existence p learned by now (recall that, given a $\sigma \in \Diamond\mathsf{STABLE}_{\leqslant N,D}(D+1)$, p does not have exact knowledge of $|\Pi_\sigma|$ and hence no *a priori* knowledge of Π_σ). Process p also holds two crucial local variables, the *lock round* ℓ and the *proposal value* x, whose significance we outline below. In order to access S in a clear way, Algorithm 1 provides the functions $\mathtt{L}(q, s)$ and $\mathtt{X}(q, s)$ that, when executed by p, return the value of ℓ resp. x of remote process q at time s, or -1 if this value is unknown to p. At the start of each round r, process p attempts to broadcast P, S, and A. By our system model, every process q with $(p, q) \in G_r$ will receive this message. Maintenance of S and A is delegated primarily to the `update` function, which merges the content of all messages received from a process q into the local copies of A and S and adds an edge (q, p), labeled with the appropriate round, to A. In addition to this, p appends its own value of ℓ and x to S at the end of round r.

Before we continue with the logic of the main algorithm, we note a key insight about our system model: To some extent, by recording received messages, interpreting them as in-edges in a communication graph, and trying to flood the system with these records, processes can reliably detect root components! Algorithm 1 implements this in the function $\mathtt{searchRoot}(s)$, which, when called by p in round r, returns a set R that satisfies two key properties: If $R \neq \emptyset$ then $R = \text{Root}(G_s)$ and if $R = \text{Root}(G_s)$ and there is a chain of messages such that $R \subseteq \text{CP}_p^r(s)$ then $\mathtt{searchRoot}(s) = R$.

The very basic operation of Algorithm 2 is to find root components and "lock on" to them, by setting the lock round ℓ to the current round and x to a deterministic function (e.g. max()) of the proposal values of the root members. In some sense, the process hopes to hit ρ of the ρ-rooted sequence of length $D + 1$ that is promised by the specification of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(D + 1)$. After this, a process simply waits for contradictory evidence that suggests that the currently locked-on root component could not have been ρ. In more detail, in each round r, Algorithm 2 proceeds in two phases:

In the first phase, p checks whether it needs to adapt its values of x or ℓ. It does this in three cases:

1. When p detects that a root component R occurred D rounds ago and p either had $\ell = 0$ or, $D + 1$ rounds ago, had detected a different root component $R' \neq R$. In either case, p sets its lock round $\ell \leftarrow r$ and its proposal x gets the maximum proposal x_q value over all processes $q \in R$ from D rounds ago in Line 9. This works because if the root component R' that was found was indeed ρ, every process must have locked on to it and thus, when a process detects a changed root component R, it is guaranteed that all members of R are already locked-on to ρ. In this way the proposal value of ρ is preserved forever.

2. If the detection from the previous case failed, p sets $\ell \leftarrow 0$ if there is evidence that contradicts the current proposal value x and lock round ℓ. That is, if a process q had $\ell_q = 0$ or a proposal different from x within the last N rounds but after the current lock round. Algorithm 1 implements this in the function $\mathtt{latestRefutation}(r - N, r - 1)$, as called in Line 13. The main aspect of this procedure is that p cannot possibly remove a lock on ρ, as ρ would lead all

processes to lock on to it and remain locked on to it forever, hence there is no contradictory evidence.

3. Possibly in addition to case 2 above, process p adapts its own proposal to v if p sees that every process that was locked on to something during the last N rounds was locked on to v. This is to ensure that processes adapt their proposal if there is a set of locked-on processes that never learned of contradictory evidence and might be tempted to assume that their lock value stems from ρ itself. In this case, the function `uniqueCandidate`$(r-N, r-1)$ returns the value v when called in Line 16.

In the second phase, process p waits until it is safe to decide on x. This is the case when, according p's point of view, in the last $N(D+2N)$ rounds all processes are locked on and have the same proposal value. Process p performs this check via the call to `allGood`$(r-N(D+2N), r-1)$ in Line 17. The crucial point here is that, by a pigeon-hole argument, the observation of $N(D+2N)$ rounds where all processes are locked on and have the same proposal value implies that there was in fact a "v-locked sequence" of $D+2N$ rounds. A v-locked sequence consists only of communication graphs where every process in the root component is locked on and has the same proposal value. Such a sequence guarantees that all future proposal values are v, thereby ensuring the safety of the algorithm.

8 Beyond rootedness and consecutive stability

So far, we have studied how the duration of a stability phase affects consensus solvability in message adversaries where every communication graph is rooted. In this section, we extend these results to investigate adversaries that do not satisfy all properties of $\Diamond\mathsf{STABLE}_{\leqslant N,D}$ but only some of those.

First, we study the case where the adversary guarantees only a constant fraction of the communication graphs to be rooted. This turns out to be a rather benign case, as our previously established techniques are almost directly applicable here. Second, we look into the case where we do not require a constant fraction of communication graphs to be rooted. For this purpose, we introduce $\mathsf{STICKY}(x)$, a message adversary that guarantees that all root components R that are vertex-stable for more than x rounds have an R-rooted subsequence with a duration $\geq x$ during their lifetime. This means that there may be many root components that are overlapping, in the sense that they are vertex-stable simultaneously, but if one of them is stable long enough, it must become the only stable root component for a certain duration. Third, we investigate the impact of the duration of non-consecutive stability. We assume that this duration is fixed to D rounds and study the solvability of consensus in relation to the parameter of x of $\mathsf{STICKY}(x)$. Finally, we introduce a consensus algorithm that shows that consensus is solvable if the

Algorithm 1: Helper functions for process p

```
Function update($q$, $P_q$, $S_q$, $A_q$):
    $P \leftarrow P \cup \{q\} \cup P_q$
    $S \leftarrow S \cup S_q$
    $A \leftarrow A \cup \{(r, q, p)\} \cup A_q$
Function searchRoot($s$):
    $V \leftarrow \{v \in P \mid \exists(s, *, v) \in A$ or $\exists(s, v, *) \in A\}$
    $E \leftarrow \left\{(u, v) \in P^2 \mid \exists(s, u, v) \in A\right\}$
    $Ret \leftarrow \emptyset$
    Let $SCC(V, E)$ denote the set of vertex sets of the strongly connected
      components (SCCs) of $\langle V, E\rangle$. A single node $q$ may constitute a SCC only
      if $(q, q) \in E$.
    foreach $C \in SCC(V, E)$ do
        if $\nexists v \in V \setminus C\colon (v, u) \in E$ for a $u \in C$ then
            $Ret \leftarrow Ret \cup \{C\}$
    return Ret
Function L($q, s$):
    if $\exists(q, s, *, \ell) \in S$ then return $\ell$
    else return $-1$
Function X($q, s$):
    if $\exists(q, s, x, *) \in S$ then return x
    else return $-1$
Function latestRefutation($a, b$):
    $T \leftarrow \{i \in [a, b] \mid \exists q \in P\colon L(q, i) = 0$ or $X(q, i) \notin \{-1, x\}\}$
    if $T \neq \emptyset$ then return $\max(T)$
    else return $-1$
Function uniqueCandidate($a, b$):
    if $\exists k \in \mathbb{N}\colon \forall u \in P, \forall i \in [a, b]\colon L(u, i) > 0 \Rightarrow X(u, i) = k$ and $\exists q \in P$,
      $\exists j \in [a, b]\colon L(q, j) > 0$ then
        return $k$
    else
        return $-1$
Function allGood($a, b$):
    return $(\forall q \in P, \forall i \in [a, b]\colon L(q, i) \neq 0$ and $X(q, i) \in \{-1, x\})$
```

above impossibility conditions are not met.

Algorithm 2: Consensus algorithm, code for process p. Uses function definitions from Algorithm 1.

```
Initialization:
$r \leftarrow 0$, $\mathsf{x} \leftarrow x_p$, $\ell \leftarrow 0$, $\mathsf{A} \leftarrow \emptyset$, $\mathsf{P} \leftarrow \emptyset$
$\mathsf{S} \leftarrow \{(p, r, \mathsf{x}, \ell)\}$
$r \leftarrow 1$

Round $r$ communication:
Attempt to send $(\mathsf{P}, \mathsf{S}, \mathsf{A})$ to all
Receive $m_q$ from all $q$ with $(q, p) \in G_r$

Round $r$ computation:
foreach $m_q$ s.t. $p$ received $m_q = (\mathsf{P}_q, \mathsf{S}_q, \mathsf{A}_q)$ in round $r$ do
    update($q, \mathsf{P}_q, \mathsf{S}_q, \mathsf{A}_q$)
$\mathsf{R} \leftarrow$ searchRoot($r - D$)
if $\mathsf{R} \neq \emptyset$ and ($\ell = 0$ or $\mathsf{R} \neq$ searchRoot($r - D - 1$)) then
    $\mathsf{x} \leftarrow \max\{\mathtt{X}(q, r - D) \mid q \in \mathsf{R}\}$
    $\ell \leftarrow r$
else if $r > N$ then
    if latestRefutation($r - N, r - 1$) $\geqslant \ell$ then
        $\ell \leftarrow 0$
    if uniqueCandidate($r - N, r - 1$) $\neq -1$ then
        $\mathsf{x} \leftarrow$ uniqueCandidate($r - N, r - 1$)
if $r > N(D + 2N)$, $y_p = \perp$, $\ell > 0$, and allGood($r - N(D + 2N), r - 1$) = TRUE
  then
    $y_p \leftarrow \mathsf{x}$
$\mathsf{S} \leftarrow \mathsf{S} \cup (p, r, \mathsf{x}, \ell)$
$r \leftarrow r + 1$
```

8.1 Dealing with a constant fraction of rooted graphs

We start our investigation with an adversary that does not satisfy the conditions of ROOTED_n in that it contains sequences where not all communication graphs are rooted graphs. Recall that this means that some communication graphs may contain multiple strongly connected components such that there is no directed path between them. In this sense, these strongly connected components define a partitioning of the system (even though the components may still be weakly connected to each other). Below we show how, depending on the severity of this partitioning, consensus may still solvable with the techniques described so far. In particular, it turns out that our methods are still applicable if there remains a constant fraction of rooted graphs in every sequence of communication graphs.

In order to precisely express this type of partitioning, we introduce the message adversary $\mathsf{ROOTED}_n(k)$.

Definition 8.1. $\mathsf{ROOTED}_n(k)$ *is the set of those infinite communication patterns* σ *where each subsequence* $\sigma' \subset \sigma$ *with* $|\sigma'| = k$ *contains at least one rooted communication graph.*

By definition, $\mathsf{ROOTED}_n(k)$ describes those graph sequences where at least every kth communication graph is rooted. For this kind of relaxation of ROOTED_n, it is actually easy to adapt Algorithm 2. The reason for this is that $\lozenge\mathsf{GOOD}_n(D+1)$ still guarantees a rooted subsequence of $D+1$ consecutive rounds and the more relaxed condition $\mathsf{ROOTED}_n(k)$ merely implies that we can rely only on every kth graph to be rooted. It follows that essentially waiting k times longer in Algorithm 2 will result in the same amount of rooted graphs and thus the same level of information propagation throughout the execution. In the code, this simply amounts to increasing all durations by a factor of k, as presented in Algorithm 3.

Regarding the impossibility results from Section 6, it is not hard to see that they hold also for $\mathsf{ROOTED}_n(k)$. The reason for this is that the adversary may simply choose completely disconnected graphs (with self-loops) in all rounds except in the stability phase and in one round of every subsequence of length k, thereby satisfying $\mathsf{ROOTED}_n(k)$. This ensures that there is no information propagation in those rounds, which makes Theorem 6.1 immediately applicable. As, furthermore, the arguments for the correctness of Algorithm 2 can be adapted to Algorithm 3, we have the following lemma.

Lemma 8.2. *Fix some* $k \in \mathbb{N}^*$. *Consensus is solvable under* $\mathsf{ROOTED}_n(k) \cap \lozenge\mathsf{GOOD}_n(x) \cap \mathsf{DEPTH}_n(D)$ *if and only if* $x > D$.

8.2 Lower bounds for partitioned graphs

Until now, we have seen that a constant fraction of rooted graphs is a very benign case of a partitioning in the sense that essentially all our previous results remain applicable. As we have motivated in the beginning, however, we would like to be able to model systems that can cope with arbitrary long periods of initial uncertainty. The question that we answer in the remainder of this section is how this can be realized with respect a partitioning that remains for an arbitrary long period. For this reason, we introduce the adversary $\mathsf{STICKY}(x)$ below. Informally, it restricts a partitioning only in that any vertex-stable root component that is stable for $\geq x$ rounds is the only root component at some point during its lifetime for at least x consecutive rounds.

Definition 8.3. *For* $x \geq 1$, $\mathsf{STICKY}(x)$ *is defined as:* $\mathsf{STICKY}(1) = \mathsf{ROOTED}_n$ *and for* $x > 1$, $\mathsf{STICKY}(x)$ *is the set of all infinite communication graph sequences* σ

Algorithm 3: Consensus algorithm for $\mathsf{ROOTED}_n(k) \cap \Diamond\mathsf{GOOD}_n(D+1) \cap \mathsf{DEPTH}_n(D)$, code for process p. Uses function definitions from Algorithm 1.

```
   Initialization:
1  $r \leftarrow 0, \mathsf{x} \leftarrow x_p, \ell \leftarrow 0, \mathsf{A} \leftarrow \emptyset, \mathsf{P} \leftarrow \emptyset$
2  $\mathsf{S} \leftarrow \{(p, r, \mathsf{x}, \ell)\}$
3  $r \leftarrow 1$

   Round $r$ communication:
4  Attempt to send $(\mathsf{P}, \mathsf{S}, \mathsf{A})$ to all
5  Receive $m_q$ from all $q$ with $(q, p) \in G_r$

   Round $r$ computation:
6  foreach $m_q$ s.t. $p$ received $m_q = (\mathsf{P}_q, \mathsf{S}_q, \mathsf{A}_q)$ in round $r$ do
7      update($q, \mathsf{P}_q, \mathsf{S}_q, \mathsf{A}_q$)
8  $\mathsf{R} \leftarrow$ searchRoot($r - D$)
9  if $\mathsf{R} \neq \emptyset$ and ($\ell = 0$ or $\mathsf{R} \neq$ searchRoot($r - D - 1$)) then
10     $\mathsf{x} \leftarrow \max\{\mathtt{X}(q, r - D) \mid q \in \mathsf{R}\}$
11     $\ell \leftarrow r$
12 else if $r > kN$ then
13     if latestRefutation($r - kN, r - 1$) $\geqslant \ell$ then
14         $\ell \leftarrow 0$
15     if uniqueCandidate($r - kN, r - 1$) $\neq -1$ then
16         $\mathsf{x} \leftarrow$ uniqueCandidate($r - kN, r - 1$)
17 if $r > kN(D + 2N)$, $y_p = \perp$, $\ell > 0$, and allGood($r - kN(D + 2N), r - 1$) = TRUE
     then
18     $y_p \leftarrow \mathsf{x}$
19 $\mathsf{S} \leftarrow \mathsf{S} \cup (p, r, \mathsf{x}, \ell)$
20 $r \leftarrow r + 1$
```

such that $|\Pi_\sigma| \leq N$ and, for every subsequence $\sigma' = (G_r)_{r=a}^{b} \subseteq \sigma$ with $|\sigma'| \geq x$, if there is a set $R \subseteq \Pi_\sigma$ such that R is a root component of every G_r with $r \in [a, b]$ but neither of G_{a-1} nor G_{b+1}, then there is an R-rooted sequence $\sigma'' \subseteq \sigma'$ with $|\sigma''| \geq x$.

By convention, $\mathsf{STICKY}(1)$ is equivalent to ROOTED_n. For $x > 1$, Definition 8.3 essentially states that every sequence of length $\geq x$ in $\mathsf{STICKY}(x)$ that consists of communication graphs in which R occurs as a root component, in fact contains an R-rooted subsequence of length x. This permits vertex-stable root components that behave almost arbitrarily chaotic if their duration is $< x$ rounds and also allows multiple root components to some extent, even if their duration is $\geq x$. It does, however, provide a very important guarantee that will be exploited algorithmically later on: The first vertex-stable root component that remains stable for $\geq x$ rounds, must become the only vertex-stable root component for a phase of $\geq x$ rounds *before* any subsequent long-lived vertex-stable root components come into existence.

As we will see below, solving consensus in systems that are partitioned in the sense of Definition 8.3 incurs a necessity for longer periods of stability than before. In order to show this, we introduce the combined message adversary $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y)$. Taken together, this lets us formulate a generalized version of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x)$, which can express partitioned communication graphs as well as non-consecutive liveness periods:

Definition 8.4. $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y) := \mathsf{STICKY}(x) \cap \Diamond\mathsf{GOOD}_n(y) \cap \mathsf{DEPTH}_n(D)$.

It is not hard to see that, because $\mathsf{ROOTED}_n = \mathsf{STICKY}(1)$ by convention, we have $\Diamond\mathsf{STABLE}_{\leqslant N,D}(1, y) = \Diamond\mathsf{STABLE}_{\leqslant N,D}(y)$.

In order to extend the characterization of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y)$ to the entire range of its parameters, we should investigate the cases where $x > 1$. When setting $x > 1$ in $\mathsf{STICKY}(x)$, however, we unfortunately cannot simply apply the techniques from Algorithm 3 and Theorem 6.1 again, but instead need to develop new algorithms and impossibility results. We will find that a significantly longer period of stability than before is required to solve consensus in this case: In Lemma 8.5 below, we find that increasing consecutive stability to $y = 2D$ still renders consensus impossible due to the partitioning that occurs when $x > 1$.

We prove this lower bound for the message adversary $\mathsf{MA} = \bigcap_{i>1} \mathsf{STICKY}(i) \cap \Diamond\mathsf{GOOD}_n(2D) \cap \mathsf{DEPTH}_n(D)$, which is sufficient since $\mathsf{MA} \subseteq \Diamond\mathsf{STABLE}_{\leqslant N,D}(x, 2D)$ for $x > 1$. Glossing over many of the details, MA may generate σ_1 and σ_4 from Figure 5, resulting in admissible executions $\varepsilon_1 = \langle C^0, \sigma_1\rangle$ and $\varepsilon_4 = \langle C^0, \sigma_4\rangle$, where C^0 is an initial configuration in which the processes of $A = \{a, a'\}$ and the processes of $B = \{b, b'\}$ start with different inputs. The validity condition implies that in any correct consensus algorithm, by some round τ, b decided a different value in ε_1 than a in ε_4. Furthermore, MA may generate σ_2 and σ_3 from Figure 5 where $\varepsilon_2 = \langle C^0, \sigma_2\rangle$ and $\varepsilon_3 = \langle C^0, \sigma_3\rangle$ result in the indistinguishabilities $\varepsilon_1 \sim_b \varepsilon_2 \sim_c \varepsilon_3 \sim_a \varepsilon_4$. This implies that a and b decided differently in ε_2 and ε_3 and hence c can never make a correct decision in ε_2 or ε_3.

Lemma 8.5. *Fix an arbitrary integer $x > 1$. Consensus is impossible under $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y)$ if $y \leq 2D$.*

8.3 Lower bounds for non-consecutive stability

As we have just seen, even under a slightly partitioned message adversary, we require already a stability phase that is about twice as long as in the case where

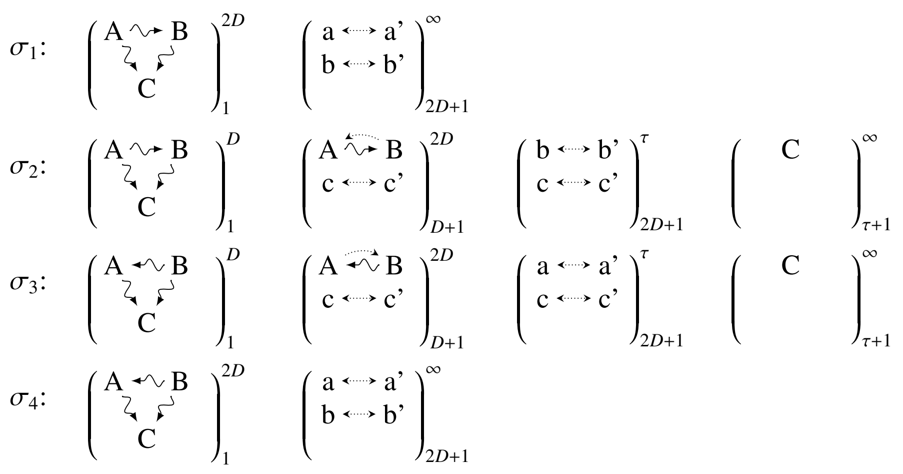


Figure 5: Sequences of Lemma 8.5. We use an upper case letter X to denote the graph $x \leftrightarrow x'$. For $D \geqslant 1$, a "wavy" edge $X \rightsquigarrow Y$ stands for a chain $X \rightarrow p_{i_1} \rightarrow \ldots \rightarrow p_{i_{D-1}} \rightarrow Y$ where $i_j \neq i_{j'}$ for $j \neq j'$. $X \rightarrow p$ (resp. $p \rightarrow X$) stands for the presence of the two edges (x, p) and (x', p) (resp. (p, x) and (p, x')). The chain is unique wrt. X and Y and its vertex-set is disjoint from that of any other chain, denoted by a "wavy" line in the same graph. For $D = 1$, $X \rightsquigarrow Y$ is simply an ordinary edge $X \rightarrow Y$. A dotted, unidirectional edge $x \dashrightarrow y$ represents an edge that is in G_i iff it is not in G_{i-1}. A dotted, bidirectional edge $x \leftarrow\!\!\!\dashrightarrow y$ means that there is a single unidirectional edge between x and y that alternates its direction in every round. We assume that there is an edge from each process (or set of processes) depicted in the graph to every process not depicted in the graph.

every communication graph is rooted. A question that arises in this context is whether this stability really needs to be consecutive, as we have required so far. In order to investigate this, i.e., to model the behavior of partly consecutive stability, we introduce a generalized variant of $\Diamond\mathsf{GOOD}_n(x)$ below.

Definition 8.6. $\Diamond\mathsf{GOOD}_n(y, z)$ *is the set of all infinite communication graph sequences* σ *such that* $|\Pi_\sigma| = n$ *and there exists a set* $R \subseteq \Pi_\sigma$ *and an* R*-rooted* $\sigma' \subseteq \sigma$ *with* $|\sigma'| \geqslant y$. *In addition, the subsequence* σ' *is followed by a finite subsequence* $\sigma'' \subseteq \sigma$ *such that at least* z *communication graphs* G_r *of* σ'' *satisfy* $\mathrm{Root}(G_r) = R$.

Definition 8.6 allows us to express that an R-rooted sequence of length at least x is followed by y not necessarily consecutive communication graphs with unique root component R. It leads to our final, most general definition of the eventually stabilizing message adversary:

Definition 8.7. $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y, z) := \mathsf{STICKY}(x) \cap \Diamond\mathsf{GOOD}_n(y, z) \cap \mathsf{DEPTH}_n(D)$.

A particularly interesting value for the second parameter z of $\Diamond\mathsf{GOOD}_n(y, z)$ seems to be $z = D$: According to Definition 5.1, this is precisely enough for the state information of the consecutively vertex-stable root component to propagate from the end of its stability phase to all other processes. Our lower bound, expressed in Lemma 8.8 below, thus states that consensus is impossible under $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y, D)$ if $y < x$. The reason is that the information propagation during a stability phase of y rounds, where $y < x$, is insufficient for some process to reliably detect that the stability window has occurred. Subsequently, this turns out to be a tight bound in the sense that consensus becomes solvable if $y \geq x$.

Lemma 8.8. *Fix an integer $x > 1$ and $y \in \mathbb{N}^*$. Consensus is impossible under $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y, D)$ if $y < x$.*

8.4 A general solution algorithm

We now show that consensus is solvable under $\Diamond\mathsf{STABLE}_{\leqslant N,D}(z_1, z_2, D)$ for $z_2 \geq z_1 > D$ by providing Algorithm 4, which copes with this message adversary, despite its power to partition the communication graphs and providing only a partially consecutive stability phase. The algorithm reuses some of the supporting functions from Algorithm 1 and employs the same mechanisms to approximate the graph sequence and store/forward the processes' initial values.

Informally, in a round r, the algorithm checks if it can reliably find a unique root component R in round $r - D$ (Lines 8 and 9). If this is the case, the algorithm searches for the first round s such that this root component consecutively occurred in $(G_i)_{i=\mathsf{s}}^{r}$ and locks the maximum proposal value $\mathsf{x}_q^{\mathsf{s}}$ a process $q \in \mathsf{R}$ held at time s (Lines 10 to 12). This ensures that if a unique root component occurs in a subsequence of at least $D + 1$ rounds, every process has the same lock value, forever after.

The criterion for a decision is straightforward in Algorithm 4: a process p decides in Line 24 as soon as it detects a R'-rooted subsequence of duration at least z_2 (Lines 19 to 22). Compared to our previous approaches, it is interesting to note that fixing the third parameter of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(z_1, z_2, D)$ to D here ensures that every process eventually must detect this subsequence, thereby ensuring termination. Again, the algorithm decides on the largest value of $\mathsf{x}_q^{\mathsf{t}}$ of $q \in \mathsf{R}'$, where t is the earliest round such that R' appeared as a root component in every communication graph G_r for $r \in [\mathsf{t}, i]$ but not in $G_{\mathsf{t}-1}$ (Line 23). Since we have that $z_2 \geq z_1$, it is guaranteed that the first sequence detected in this way is locked by all processes forever after, according to the mechanism described in the previous paragraph. It follows that all future decisions are the same.

Taken together with our previous possibility and impossibility results, this enables a characterization of the consensus solvability of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y, D)$ for

Algorithm 4: Consensus algorithm for $\Diamond\mathsf{STABLE}_{\leqslant N,D}(z_1, z_2, D)$, code for process p. Uses function definitions from Algorithm 1.

```
Initialization:
r ← 0, x ← $x_p$, ℓ ← 0, A ← ∅, P ← ∅
S ← {(p, r, x, ℓ)}
r ← 1

Round r communication:
Attempt to send (P, S, A) to all
Receive $m_q$ from all q with $(q, p) \in G_r$

Round r computation:
foreach $m_q$ s.t. p received $m_q = (P_q, S_q, A_q)$ in round r do
    update($q, P_q, S_q, A_q$)
R ← searchRoot($r - D$)
if |R| = 1 then
    s ← $r - D$
    while s > 1 and R ⊆ searchRoot(s − 1) do s ← s − 1
    x ← max{X(q, s) | q ∈ R}
R′ ← ∅
i ← 0
while $y_p = \bot$ and i < r − 1 do
    i ← i + 1
    if |searchRoot(i)| ≠ 1 then R′ ← ∅
    else // |searchRoot(i)| = 1
        if searchRoot(i) ≠ R′ then
            R′ ← searchRoot(i)
            t ← i
        if i − t + 1 ≥ $z_2$ then
            while t > 1 and R′ ⊆ searchRoot(t − 1) do t ← t − 1
            $y_p$ ← max{X(q, t) | q ∈ R′}
S ← S ∪ (p, r, x, ℓ)
r ← r + 1
```

all values of $x, y \in \mathbb{N}^*$, which is shown in Fig. 6.

9 Additional Results

In addition to what we presented so far, [33] develops some additional results that we will sketch briefly below.

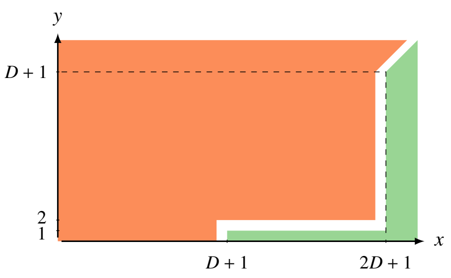


Figure 6: Characterization of the consensus solvability/impossibility border of $\Diamond\mathsf{STABLE}_{\leqslant N,D}(x, y, z)$, where $z = 0$ if $y \leq D + 1$ and $z = D$ otherwise: red (upper left) = impossible, green (right) = possible.

Strongest message adversary A popular method to model consensus solvability in distributed computing systems is the failure detector formalism. In essence, and without going into the details, a failure detector is an oracle that can be queried by a process to yield additional information it might not have obtained otherwise and which helps the process to solve consensus. Traditionally, this formalism is used in distributed computing systems prone to crash failures, where the output of the failure detector is some function on the processes that have crashed so far. This makes sense, since, in these systems, the crucial difficulty is to locally distinguish a crashed process from a silent one (which could occur, e.g., in an asynchronous system because a process is very slow). Perhaps the most celebrated contribution of the failure detector formalism is the introduction of a hierarchy among the failure detectors that allows to compare failure detectors to each other using a simulation procedure. This leads to the definition of a weakest failure detector for consensus, which should encapsulate exactly the information needed to achieve consensus solvability in a system where consensus is ordinarily impossible.

We investigate how this idea can be transferred to the message adversary model, in order to establish an order relation among the message adversaries, using an adaptation of the simulation procedure from the failure detector formalism. On the positive side, we are able to identify a class of strongest message adversaries that still enable a solution for consensus. On the negative side, we discover that this class is rather large, which means that the granularity of such a simulation approach is not very satisfactory. A preliminary version of this work appeared in [28].

***k*-set agreement** A natural follow-up question to Section 8 is whether systems that do not allow consensus solutions because they do not guarantee an adequate replacement for rootedness allow solutions to easier problems than consensus, such as k-set agreement. The k-set agreement problem is a generalization of consensus where not all decision values must be the same but instead there may be at most k different decision values. Perhaps surprisingly though, even if the number of rooted spanning trees in each communication graph is limited to k, k-set agreement is still impossible. In fact, it turns out that even more severe restrictions than this render k-set agreement impossible. In order to make k-set agreement solvable, it seems that there needs to be some kind of "significant influence" relation from a set of k select vertex-stable root components to all subsequent vertex-stable root components. One instance of such a k-set agreement algorithm was presented in [7].

Limit-closed message adversaries Regarding a complete characterization of consensus solvability, we show that a simple characterization exists for the class of limit-closed message adversaries. Limit-closed means that if the limit of every sequence $(\sigma_r)_{r>0}$, where σ_r is an admissible communication pattern, whose first r rounds coincide with the first r rounds of σ_{r+1}, is admissible. Our characterization shows that consensus is solvable under a limit-closed message adversary, if and only if, every admissible communication pattern σ has a round r and a process q such that the following holds: in all communication patterns that are transitively indistinguishable to σ in round r, all processes have heard from q, i.e., for every $p \in \Pi$ we have $q \in \mathrm{CP}_p^r(0)$.

Point set topology Going one step further, we look for potential topologies for the set of process-time pairs, which result from the product of the initial configurations with the communication patterns. We find that there exists a semi-metric that expresses distance in certain terms of indistinguishability, which can be used to define a semi-metric, topological space on the process-time pairs on which a consensus characterization can be formulated: Exploiting the well-known fact that the consensus task is a continuous map, it can be shown that consensus solvability is equivalent to the existence of a separation of this space that respects the validity condition. In other words, we show that for every input value v, the process time pair with the initial configuration where all processes start with v cannot be in the same connected component as the process time pair with the initial configuration where all processes start with $v' \neq v$.

References

[1] IEEE 802.11 standard: Wireless lan medium access control (mac) and physical layer (phy) specifications, June 2007. IEEE Computer Society LAN MAN Standards Committee.

[2] Y. Afek and E. Gafni. Asynchrony from synchrony. In D. Frey, M. Raynal, S. Sarkar, R. Shyamasundar, and P. Sinha, editors, *14th International Conference on Distrib-*

uted Computing and Networking (ICDCN), volume 7730 of *LNCS*, pages 225–239. Springer Berlin Heidelberg, 2013.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. John Wiley & Sons, 2nd edition, 2004.

[4] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 7355 of *LNCS*, pages 73–84. Springer Berlin Heidelberg, 2012.

[5] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. In A. Bouajjani and H. Fauconnier, editors, *Third International Conference on Networked Systems (NETYS)*, pages 109–124. Springer International Publishing, 2015.

[6] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science*, 726:41 – 77, 2018.

[7] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *Theoretical Computer Science*, 726:41–77, 2018.

[8] M. Biely, U. Schmid, and B. Weiss. Synchronous consensus under hybrid process and link failures. *Theoretical Computer Science*, 412(40):5602 – 5630, 2011.

[9] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11(3):420 – 440, 1990.

[10] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[11] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In M. M. Halldòrsson, K. Iwama, N. Kobayashi, and B. Speckmann, editors, *42nd International Colloquium on Automata, Languages, and Programming*, volume 9135 of *Lecture Notes in Computer Science*, pages 528–539. Springer Berlin Heidelberg, 2015.

[12] B. Charron-Bost and A. Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, Apr. 2009.

[13] É. Coulouma, E. Godard, and J. G. Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theoretical Computer Science*, 584:80–90, 2015.

[14] T. Fevat and E. Godard. Minimal obstructions for the coordinated attack problem and beyond. In *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16-20 May, 2011 - Conference Proceedings*, pages 1001–1011, 2011.

[15] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[16] E. 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*, pages 143–152, Puerto Vallarta, Mexico, 1998. ACM Press.

[17] A. Goiser, S. Khattab, G. Fassl, and U. Schmid. A new robust interference reduction scheme for low complexity direct-sequence spread-spectrum receivers: Performance. In *Proceedings 3rd International IEEE Conference on Communication Theory, Reliability, and Quality of Service (CTRQ'10)*, pages 15–21, Athens, Greece, June 2010.

[18] I. Keidar and A. Shraer. Timeliness, failure detectors, and consensus performance. In *Proceedings of the twenty-fifth annual ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing (PODC'06)*, pages 169–178, New York, NY, USA, 2006. ACM Press.

[19] W. Kiess and M. Mauve. A survey on real-world implementations of mobile ad-hoc networks. *Ad Hoc Networks*, 5(3):324–339, Apr. 2007.

[20] F. Kuhn, N. A. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing (STOC)*, pages 513–522, 2010.

[21] F. Kuhn, R. Oshman, and Y. Moses. Coordinated consensus in dynamic networks. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC)*. ACM, 2011.

[22] F. Legendre, T. Hossmann, F. Sutton, and B. Plattner. 30 years of wireless ad hoc networking research: What about humanitarian and disaster relief solutions? What are we still missing? In *International Conference on Wireless Technologies for Humanitarian Relief (ACWR)*. IEEE, 2011.

[23] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, New York, NY, USA, 2004. ACM.

[24] D. Pfleger and U. Schmid. A framework for connectivity monitoring in wireless sensor networks. In *Proceedings 10th International Conference on Sensor Technlogies and Applications (SENSORCOMM)*, pages 40–48. IARIA, 2016. `https://www.thinkmind.org/download.php?articleid=sensorcomm_2016_3_10_10013`.

[25] M. Raynal and J. Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 166–175, 2013.

[26] N. Santoro and P. Widmayer. Time is not a healer. In B. Monien and R. Cori, editors, *6th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 304–313, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[27] U. Schilcher, C. Bettstetter, and G. Brandner. Temporal correlation of interference in wireless networks with rayleigh block fading. *IEEE Transactions on Mobile Computing*, 11(12):2109–2120, 2012.

[28] U. Schmid, M. Schwarz, and K. Winkler. On the strongest message adversary for consensus in directed dynamic networks. In Z. Lotker and B. Patt-Shamir, editors, *Structural Information and Communication Complexity*, pages 102–120, Cham, 2018. Springer International Publishing.

[29] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912–1951, 2009.

[30] M. Schwarz, K. Winkler, and U. Schmid. Fast consensus under eventually stabilizing message adversaries. In *Proceedings of the 17th International Conference on Distributed Computing and Networking (ICDCN)*, pages 7:1–7:10. ACM, 2016.

[31] J.-P. Sheu, C.-M. Chao, and C.-W. Sun. A clock synchronization algorithm for multihop wireless ad hoc networks. In *24th International Conference on Distributed Computing Systems (ICDCS)*, pages 574–581, 2004.

[32] C. Ware, J. Judge, J. Chicharo, and E. Dutkiewicz. Unfairness and capture behaviour in 802.11 adhoc networks. In *2000 IEEE International Conference on Communications. ICC 2000. Global Convergence Through Communications.*, 2000.

[33] K. Winkler. *Solvability of Consensus under Message Adversaries*. PhD thesis, TU Wien, 2019. To appear.

[34] B. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.