FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Solving k-Set Agreement in Dynamic Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Technische Informatik

eingereicht von

Manfred Schwarz
Matrikelnummer 0725898

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ulrich Schmid
Mitwirkung: -

Wien, August 28, 2013

(Unterschrift VerfasserIn) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Solving k-Set Agreement in Dynamic Networks

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computer Engineering

by

Manfred Schwarz
Registration Number 0725898

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr. Ulrich Schmid
Assistance: -

Vienna, August 28, 2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Manfred Schwarz
Adalbert-Stifter Straße 22 Tür 37/38, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)

(Unterschrift Verfasserin)

Acknowledgements

First and foremost I want to thank my parents, Kurt and Maria Schwarz, for their unconditional support over the last few years.

Then I would like to thank my advisor Univ.Prof. Ulrich Schmid for giving me the opportunity for this thesis and the constant support in every aspect of my work.

Also, I want to thank my fellow student Kyrill Winkler for the enriching talks, which helped me out in a few tight spots during the creation of my thesis.

I want to thank Peter Robinson and Martin Biely, as they have proof-read the publication, which was created in parallel to this work and addresses the same topic, thus they also have indirectly proof-read this master thesis.

Furthermore my girlfriend Tina Pruckermayr for sticking with me the last years and supporting my work at the university.

Lastly, the financial support by the Austrian Science Fund (FWF) project RiSE (S11405) is also gratefully acknowledged.

Abstract

In this Master thesis, we will propose a way to solve k-set agreement in a very relaxed model of a synchronous distributed system. The model is based on a sequence of directed communication graphs determined by the adversary, which specify which process receives a message from which process in a round. k-set agreement describes the problem where distributed process with possible different initial values have to agree on a smaller set of decision values. In order to make k-set agreement solvable, the power of the adversary w.r.t. disrupting communication between processes must be restricted. Essentially, the proposed restrictions ensure that, during every execution, certain strongly connected subgraphs (vertex stable root components) are influencing each other and that, eventually, communication graphs are sufficiently stable for a short time to ensure termination. We will show that an algorithm exists, which can solve k-set agreement under these weak constraints, and prove its correctness. Basically, the algorithm locally estimates whether a vertex stable root component has formed and, if so, tries to use its strong connectivity to force all its members to decide on the same value. The algorithm is k-uniform, i.e., independent of k, hence automatically adapts the number of different decision values according to the actual network conditions. The result of this work, which has been supported by Austrian Science Fund (FWF) project RiSE (S11405), have been submitted to the 17th International Conference on the Principles of Distributed Systems (OPODIS 2013).

Kurzfassung

In dieser Diplomarbeit stelle ich eine Lösung des “k-set agreement“-Problems unter einem schwachen, verteilten, synchronen Berechnungsmodel vor. Das Model basiert auf einer Sequenz von gerichteten Kommunikationsgraphen, die, vom Adversary bestimmt, vorgibt, welcher Prozess von welchem Prozess pro Runde eine Nachricht erhält. “k-set agreement“ beschreibt das Problem, in dem Prozesse mit möglicherweise verschiedenen Startwerten sich auf ein kleineres Set von Entscheidungswerten einigen müssen. Um “k-set agreement“ lösbar zu machen, müssen die Möglichkeiten des Adversaries zur Unterbindung der Kommunikation zwischen Prozessen eingeschränkt werden. Diese Restriktionen stellen sicher, dass in jeder Ausführung stark verbundene Strukturen (sogenannte “vertex stable root components“) sich gegenseitig beeinflussen und irgendwann ausreichend lang stabil sind, um Termination zu garantieren. Wir geben einen Algorithmus an, der unter diesen Einschränkungen “k-set agreement“ löst und beweisen seine Korrektheit. Der Algorithmus versucht, lokal “vertex stable root components“ zu erkennen und diese zu nutzen, um alle darin enthaltenen Prozesse zu einer gemeinsamen Entscheidung zu bringen. Der Algorithmus ist k-uniform, also unabhängig von k, und passt daher die Anzahl unterschiedlicher Entscheidungswerte automatisch an die aktuellen Bedingungen im Netzwerk an. Die Resultate dieser Arbeit, die Unterstützung durch das Projekt RiSE (S11405) des österreichischen Fonds zur Förderung wissenschaftlicher Forschung (FWF) erhielt, wurden bei der 17. International Conference on the Principles of Distributed Systems (OPODIS 2013) eingereicht.

Contents

CHAPTER 1

Dynamic networks

1.1 Introduction

With the emergence of peer-to-peer (P2P) networks, (wireless) sensor networks, mobile adhoc networks and vehicular area networks, classic computer network properties have changed drastically. The two main differences to a normal wired network, as mentioned in [10], are (i) a large set of participants (they will be called processes in the rest of the thesis), which are unknown and may or may not change over time, and (ii) the lack of a central control instance. As a consequence, existing models and solutions do not fit a growing number of network settings.

On the other hand, such systems are particularly in need of algorithms which solve distributed computing problems like consensus or k-set agreement to compensate the missing central control. Hence models and algorithms are needed for solving agreement problems in dynamic networks.

Especially demanding and very popular examples of dynamic networks are wireless networks. In particular, due to the typical broadcast behavior and the possibility of uncontrollable receiver failures of each process, the network environment is highly dynamic and may even be unknown in certain areas. Whereas, wireline networks can adequately be modeled by a fixed pool of participants and bidirectional links in between, this is hence not realistic in wireless networks: In a wireless setting, interference phenomenon as mentioned in [24] are local effects that only affect the receiver of a link. The receiver of the reversed link is usually at a different location and hence affected by different interference scenarios.

A model that is suitable for such systems hence needs the abstraction of a pair of unidirectional links between processes, which are independent of each other. These links can fade and come into existence arbitrarily due to communication failures of departing and joining processes, for example.

Obviously, without any restriction of the adversary (who is considered responsible for determining which processes can communicate with each other), no non trivial distributed computing problem can be solved: After all, it could just prevent any communication in the system. In addition, more difficult distributed computing problems usually require more restrictions of the adversary w.r.t. its ability to prevent communication between processes.

For consensus, which is the problem of having all processes agreeing on a common decision value based on local input values, a suitable model has been proposed in [10]. The model is synchronous, in the sense that the execution of the processes develops in a sequence of lock-step rounds. In every round, all processes send (broadcast) a message to each other, receive a message from all neighbors, and perform a state transition. Although the applicability of a synchronous model is limited to systems equipped with synchronized clocks, we note that this is not an unrealistic requirement, neither in wireline [33] nor in wireless [26] networks. The model of [10] uses a sequence of communication graphs, one per round, determined by the adversary, to specify which process receives a message from which process in a round. The set of processes is static, i.e., there is no churn, but its size is possible unknown.

Unfortunately, consensus requires strong guarantees from the adversary: In each round, the network must at least be weakly connected and may contain at most one root component (a strongly connected component without incoming links), which eventually must become vertex-stable (consist of the same processes, with possibly varying internal topology) for a short period of time. Both assumptions are unlikely to hold in a wireless setting.

In this thesis, we will consider the more general problem of k-set agreement in dynamic networks. For some integer $k \geq 1$ k-set agreement requires that processes agree on at most k different decision values system-wide. Note that consensus is equivalent to 1-set agreement. Using the basic model of [10], we will propose some very weak constraints for the adversary which make k-set agreement possible: We will present a suitable algorithm and prove its correctness. Moreover, resorting to impossibility results established in collaboration with a complementary Master thesis [42], we will demonstrate that the constraints underlying our model are very close to the solvability border.

The work of this thesis has been supported by the Austrian Science Fund (FWF) project RiSE (S11405); the results haven been submitted to OPODIS'13 [39].

The thesis is divided in five chapters, the first representing an overview of different models for distributed systems and formulating a framework that allows to compare and classify those. The second chapter gives an in depth description of the model used in this thesis and a formal definition of the k-set agreement problem. In the third chapter, an algorithm is presented that solves the problem under the given model, along with a detailed proof that shows its correctness. The fourth chapter presents two impossibility results, which show that our model assumptions are very close to the solvability/unsolvability border. The thesis is concluded in chapter five with a short summary of our accomplishments and a discussion of our findings.

1.2 History and classification of dynamic networks

As briefly mentioned in the introduction, networked systems underwent a lot of changes. Different network properties where adapted to fit new emerging needs over the years. Naturally scientists working in the fields of distributed systems invented new models to capture those new features. This extensive research lead to a lot of interesting insights. Important concepts have been identified and formally defined by different researchers in different ways even though they are the same.

This inspired the authors of [17] to introduce a categorization of the different models and concepts used in distributed systems. In the following paragraphs, we will cite large parts of their work to give an overview of the different categories they identified, and later on make a connection to our model.

First, [17] distinguishes 3 main categories of existing networks, which are abstracted by different models in distributed systems.

(a) The study of communication in highly dynamic networks, e.g., broadcasting and routing in delay-tolerant networks. Such networks are characterized by highly dynamic communication links between a fixed set of participants.

(b) The exploitation of passive mobility, e.g., the opportunistic use of transportation networks. Contrary to (a), in such systems, the participants are the dynamic element.

(c) The analysis of complex real-world networks, ranging from neuroscience or biology to transportation systems or social networks, e.g., the characterization of the interaction patterns emerging in a social network. This category is characterized by special emerging properties during the execution of a system.

(a) Delay-tolerant networks

Such networks are highly dynamic and have more or less no infrastructure. Their key characteristic is the absence of guaranteed communications routes between two participants at any time instant. Examples are satellites, pedestrian and vehicular networks. These networks are often called disruption-tolerant, challenged, or opportunistic networks. Even though connectivity assumptions do not hold in general, different mechanisms are available, for example, to broadcast information. Indeed a lot of research has been invested to find new techniques to solve more advanced problems in such networks. This includes pro-active knowledge on the network schedule, delay-based optimization, encounter-based choices, or even analytical and probabilistic strategies. In all those models time is of crucial importance, thus most common graph concepts were extend by a temporal vision.

(b) Opportunistic-mobility networks

Such networks exploit the delay-tolerant network created by mobile carriers equipped with short-ranged-radio transmitters. These carriers are used for performing tasks possibly external and extraneous to the carriers. Entities like code or information can move on the carrier network, using the mobility of the carriers (sometimes called ferries). Examples are, as mentioned in [17], Cabernet, deployed in taxis in the Boston area, which delivers messages and files to users in cars, or UMas DieselNet deployed to 40 buses via WIFI nodes in Amherst. In this context, ferries following deterministic periodic trajectories are of utmost interest. This includes public transport, low earth orbiting satellites or security guard tours. Even though routing is the central aspect here, some research has been done on algorithmic network exploration, i.e, for creating a broadcast infrastructure. Again the time parameter plays an important role in the concepts and the appertaining solutions.

(c) Real world complex networks

As stated in [17], the main problem is to define and formulate mathematical models that properly abstract properties of real dynamic networks. A fundamental idea is to endow the edges with some kind of temporal information. Thus, graph properties can be studied when nodes and edges have temporal constraints. There are many different research papers listed in [17], which used such a concept and established results in different areas of computer science. To conclude, as [17] mentions, "As these investigations indicate, temporal concerns are an integral part of recent research efforts in complex systems. It is also apparent that the emerging concepts are in essence the same as those from the field of communication networks, involving again temporal definitions of the notions of paths, distance, and connectivity, as well as many higher concepts that we identify in this paper."

Time-Varying Graphs

We will use the framework of [17] to formalize a basic, abstract model for real networks. It is general enough to cover most existing models, which emerged independently in various areas of computer science.
The **time-varying graph** $TVG = (V, E, T, \rho, \zeta)$ is the core element of this framework and is defined by the following parameters:

- V is the set of all available nodes in the system
- E is the set of all available unidirectional edges in the system
- $T = \mathbb{R}$ and adds a time dimension to the system
- ρ is an indicator function defined on the cross product of E and T
- ζ is a measurement of the delay that different edges may induce

The basic graph, which is also called **underlying graph**, is defined by $G = (V, E)$. G can be seen as a footprint of the TVG. G ignores the time dimension, thus T, ρ and ζ are not relevant for G. Later we will provide a more precise and formal definition. Important to notice is that neither G nor TVG has to be connected; moreover, even if G is connected, it does not imply any connectivity properties of TVG.

Naturally one can define a subgraph TVG', which can be a subset of any part of the original TVG, be it a graph containing only a node subset V' or even the set of all edges labeled by a time subset T', for example $T' = \{1, 5, 7\}$.

Point of View

Until now we introduced a general model that can abstract a broad class of network problems. Depending on the problem it can be important to analyze the evolution from different angles. [17] defines 3 major points of view;

- view of a given relation (edges)
- view of an entity (nodes)
- the view of the global system (entire graph)

If one starts from an edge standpoint, the evolution is represented by the availability and latency of each edge between two different entities. The availability is defined by $I(e)$, which is the union of all dates at which the edge is available, more precisely $I(e) = \{t \in T : \rho(e,t) = 1\}$. The node point of view can be seen as a changing neighborhood per node. Each node thus identifies the network by neighborhood relations or maybe even transitive neighborhood relations over time.

The graph-centric evolution or the so called characteristic dates of TVG ([17]) is a sequence of snapshots of the dynamic network. It primarily corresponds to the appearance or disappearance of edges in the system. Thus the evolution of TVG can be described by a sequence S_{TVG} of graphs $G_i = (V, E_i)$ where E_i is the set of edges with $\rho(E, t_i) = 1$. Note that one can also give a formal definition of G based on G_i namely, $G = \bigcup G_i$.

Journey and Distance

A sequence J of couples $\{(e_1, t_1), (e_2, t_2)(e_3, t_3)...(e_k, t_k)\}$ such that the edges $e_1, e_2, e_3....e_k$ form a path in G is called a journey in TVG if $\rho(e_i, t_i) = 1$ and $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ for all $i \leq k$. Let us denote the starting and arrival date of J by $departure(J)$ and $arrival(J)$. The temporal length can by defined by $arrival(J) - departure(J)$. J^*_{TVG} is the set of all journeys in TVG and $J^*_{u,v}$ are all journeys starting in u and ending in v. If at least one journey in $J^*_{u,v}$ exists then we say that u can reach v, which is represented by the notation $u \rightsquigarrow v$.
The distance between nodes can be measured in two ways. Either by

- the already established temporal distance $arrival(J) - departure(J)$
- or the topological distance, which simply counts the hops from u to v in G

TVG classes

In this section, which is copied from [17] for the purpose of having a very precise definition of the different classes, we discuss the impact of properties of TVGs on the feasibility and complexity of distributed computing problems, reviewing and unifying existing work from the literature. In particular, we identify a hierarchy of classes of TVGs based on temporal properties that are formalized using the concepts presented in the previous section. These class-defining properties, organized in an ascending partial order of assumptions (see Figure 1.1), from more specific to more general, are important in that they imply necessary conditions and impossibility results for distributed computations. Let us start with the simplest Class 1, i.e., the one with the weakest assumption on the TVG.

Class 1 $\exists u \in V : \forall v \in V, u \rightsquigarrow v$
That is, at least one node can reach all the others. This condition is necessary, for example, for broadcast to be feasible from at least one node.

Class 2 $\exists u \in V : \forall v \in V, v \rightsquigarrow u$
That is, at least one node can be reached by all the others. This condition is necessary to be able to compute a function whose input is spread over all the nodes, with at least one node capable of generating the output. Any algorithm for which a terminal state may depend on all the nodes initial states also falls in this category, such as leader election or counting the number of nodes.

Class 3 (Connectivity over time): $\forall u, v \in V, u \rightsquigarrow v$
That is, every node can reach all the others; in other words, the TVG is connected in T. By the same discussions as for Class 1 and Class 2, this condition is necessary to enable broadcast from any node, to compute a function whose output is known by all the nodes, or to ensure that every node has a chance to be elected.

These three basic classes were used e.g. in [15] to investigate how relations between TVG properties and feasibility of algorithms could be formally established, based on a combination of evolving graphs [21] and graph relabellings [31]. Variants of these classes can be found in recent literature, e.g. in [25], where the assumption that connectivity over time eventually takes place among a stable subset of the nodes is used to implement failure detectors in dynamic networks.

Class 4 (Round connectivity): $\forall u, v \in V, \exists J_1 \in J^*_{(u,v)}, \exists J_2 \in J^*_{(v,u)} : arrival(J_1) \leq departure(J_2)$
That is, every node can reach all the others and be reached back afterwards. Such a condition may be required e.g. for adding explicit termination to broadcast, election, or counting algorithms.

The classes defined so far are in general relevant in the case that the lifetime is finite and a limited number of topologically changes are considered. When the lifetime is infinite, connectivity over time is generally assumed on a regular basis, and more elaborate assumptions can be considered.

Class 5 (Recurrent connectivity): $\forall u, v \in V, \forall t \in T, \exists J \in J^*_{(u,v)} : departure(J) > t$
That is, at any point t in time, the temporal subgraph $TVG_{[t,+\infty)}$ remains connected over time. This class is implicitly considered in most works on *delay-tolerant networks*. It indeed represents those DTNs where routing can always be achieved over time. This class was referred to as eventually connected networks by Awerbuch and Even in [5], although the terminology was also used with different meaning in the recent literature (which we mention in another definition below). As discussed in Section 4.1, the fact that the underlying graph $G = (V, E)$ is connected

does not imply that the TVG is connected in T, but how the snapshots G_i and thus S_{TVG} is formed matters.

Such a condition on the connectivity of the TVG is, however, necessary to allow connectivity during T and thus to perform any type of global computation. Therefore, the following three classes explicitly assume that the underlying graph G is connected.

Class 6 (Recurrence of edges): $\forall e \in E, \forall t \in T, \exists t' > t : \rho(e, t') = 1$ **and** G **is connected**

That is, if an edge appears once, it appears infinitely often. Since the underlying graph G is connected, Class 6 is a subclass of Class 5. Indeed, if all the edges of a connected graph appear infinitely often, then there must exist, by transitivity, a journey between any pairs of nodes infinitely often. In a context where connectivity is recurrently achieved, it becomes interesting to look at problems where more specific properties of the journeys are involved, e.g. the possibility to broadcast a piece of information in a shortest, foremost, or fastest manner . Interestingly, these three declinations of the same problem have different requirements in terms of TVG properties. It is, for example, possible to broadcast in a foremost fashion in Class 6, whereas shortest and fastest broadcasts are not possible (for a explanation of the different broadcast settings see [16]). Shortest broadcast becomes, however, possible if the recurrence of edges is bounded in time, and the bound is known to the nodes, a property characterizing the next class:

Class 7 (Time-bounded recurrence of edges): $\forall e \in E, \forall t \in T, \exists t' \in [t, t + \Delta), \rho(e, t') = 1$**, for some** $\Delta \in T$ **and** G **is connected**
Some implications of this class include a temporal diameter that is bounded by $\Delta Diam(G)$, as well as the possibility for the nodes to wait a period of Δ to discover all their neighbors (if Δ is known). The feasibility of shortest broadcast follows naturally by using a Δ-rounded breadth-first strategy that minimizes the topological length of journeys.
A particular important type of bounded recurrence is the periodic case:

Class 8 (Periodicity of edges): $\forall e \in E, \forall t \in T, \forall k \in \mathbb{N}, \rho(e, t) = \rho(e, t + k_p)$**, for some** $p \in \mathbb{T}$ **and** G **is connected**
The periodicity assumption holds in practice in many cases, including networks whose entities are mobile with periodic movements (satellites, guards tour, subways, or buses). The periodic assumption within a delay-tolerant network has been considered, among others, in the contexts of network exploration and routing (see [17] for additional references). Periodicity enables also the construction of foremost broadcast trees that can be re-used (modulo p in time) for subsequent broadcasts (whereas the more general classes of recurrence requires the use of a different tree for every foremost broadcast). More generally, the point in exploiting TVG properties is to rely on invariants that are generated by the dynamics (e.g. recurrent existence of journeys, periodic optimality of a broadcast tree, etc.). In some works, particular assumptions on the network dynamics are made to obtain invariants of a more classical nature. Below are some examples of classes, formulated using the graph-centric point of view of (discrete-time) evolving graphs, i.e., where $TVG = (G, S_{TVG}, \mathbb{N})$.

Class 9 (Constant connectivity): $\forall G_i \in S_{TVG}, G_i$ is connected
Here, the dynamics of the network is not constrained as long as it remains connected in every time step. Such a class was used, for example, in [34] to enable progression hypotheses on the broadcast problem. This class was also considered in [29] for the problem of consensus.
Indeed, if the network is always connected, then at every time step there must exist an edge between an informed node and a non-informed node, which allows to bound broadcast time by $n = |V|$ time steps (worst case scenario).

Class 10 (T-interval connectivity): $\forall i \in \mathbb{N}, \exists T \in \mathbb{N}, \exists G' \subseteq G : V_{G'} = V_G, G'$ is connected, and $\forall j \in [i, i+T-1), G' \subseteq G_j$
This class is a particular case of constant connectivity in which the same spanning connected subgraph of the underlying graph G is available for any period of T consecutive time steps. It was introduced in [51] to study problems such as counting, token dissemination, and computation of functions whose input is spread over all the nodes (considering an adversarial edge schedule). The authors show that computation could be sped up by a factor of T compared to the 1-interval connected graphs, that is, graphs of Class 9.
Other classes of TVGs can be found in [35] , based on intermediate properties between constant connectivity and connectivity over time. They include Class 11 and Class 12 below.

Class 11 (Eventual instant-connectivity): $\forall i \in \mathbb{N}, \exists j \in \mathbb{N} : j \geq i, G_j$ is connected. In other words, there is always a time from which on the network is connected
This class was simply referred to as eventual connectivity in [62], but since the meaning is different than that of Awerbuch and Even (connectivity over time), we renamed it to avoid ambiguities.

Class 12 (Eventual instant-routability): $\forall u, v \in V, \forall i \in N, \exists j \in N : j \geq i$ and a path from u to v exists in G_j
That is, for any two nodes, there is always a future time step in which a path exists between them. The difference to Class 11 is that these paths may occur at different times for different pairs of nodes. Classes 11 and 12 were used in [35] to represent networks where routing protocols for (connected) mobile ad hoc networks eventually succeed if they tolerate transient faults.
Most of the works listed above strove to characterize the impact of various temporal properties on problems or algorithms. A reverse approach was considered by Angluin et al. in the field of population protocols [4, 3], where for a given assumption (that any pair of node interacts infinitely often), they characterized all the problems that could be solved in this context. The corresponding class is generally referred to as that of (complete) graph of interaction.

Class 13 (Complete graph of interaction): The underlying graph $G = (V, E)$ is complete, and $\forall e \in E, \forall t \in T, \exists t' > t : \rho(e, t') = 1$
From a time-varying graph perspective, this class is the specific subset of Class 6, in which the underlying graph G is complete. Various types of schedulers and assumptions have been subsequently considered in the field of population protocols, adding further constraints to Class 13 (e.g. weak fairness, strong fairness, bounded, or k-bounded schedulers) as well as interaction

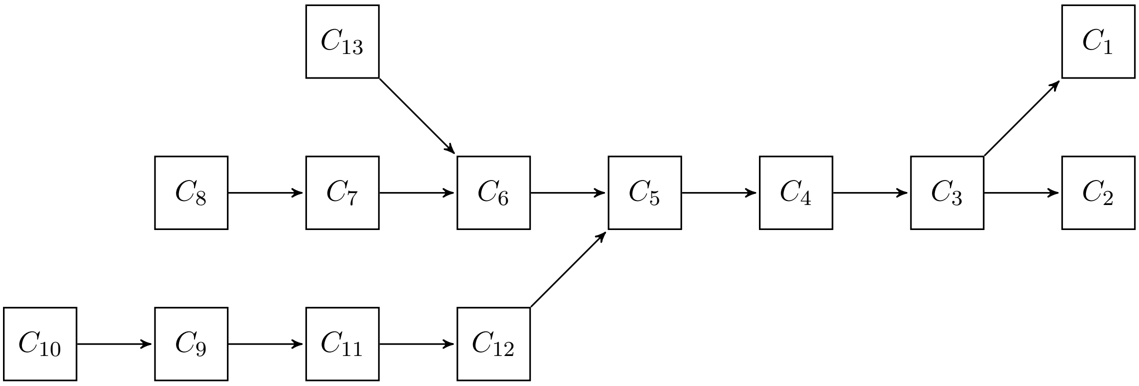


Figure 1.1: Relations of inclusion between classes (from specific to general). Arrows represent "stronger as": e.g., C_{10} is stronger as C_9, i.e., C_{10} includes C_9

graphs which might not be complete.

An interesting aspect of unifying these properties within the same formalism is the possibility to see how they relate to one another, and to compare the associated solutions or algorithms. An overview can be gained by looking at the classification shown in Figure 1.1, where basic relations of inclusion between the above classes are reported. All inclusions represented by arrows are strict: for each relation, the parent class (start point of an edge) contains some time-varying graphs that are not in the child class (end point of an edge). Clearly, one should try to solve a problem in the most general context possible. The right-most classes are so general that they offer little properties to be exploited by an algorithm, but some intermediate classes, such as Class 5, appear quite central in the hierarchy. This class indeed contains all the classes where significant work was done. A problem solved in this class would therefore apply to virtually all the contexts considered heretofore in the literature.

Such a classification may also be used to categorize problems themselves. As mentioned above, shortest broadcast is not generally achievable in Class 6, whereas foremost broadcast is. Similarly, it was shown in [14] that fastest broadcast is not feasible in Class 7, whereas shortest broadcast can be achieved with some knowledge. Since $Class7 \subset Class6$,

$foremostBcast \preceq shortestBcast \preceq fastestBcast$,

defines a partial order on these problems according to topological requirements, where A $\preceq$ B means that A is a less demanding problem than B.

Later on, we will use this framework to classify our own model and show that it belongs to very weak classes in Figure 1.1.

1.3 Related work

We are not aware of any previous work on k-set agreement in a directed and dynamic network environment with connectivity requirements as weak as ours. Nevertheless, in this section, we will present an overview of existing research that is related to at least one aspect of our solution.

Dynamic networks have been studied intensively in distributed computing. Early publications on this topic include [1] or [6].

One aspect of dynamic networks that can be used to categorize research in this area is whether the the set of processes is assumed to be fixed or subject for change. Since it would even be difficult to give a meaningful definition for k-set agreement in dynamic networks with churn [27] and node mobility [41], we will only consider static (but possibly unknown) processes.

Dynamic network research

[1] introduces a model, where the nodes and communication links of a network form a communication graph. Each edge represents an undirected link, which in turn consists of a directed pair of links. Moreover edges have a bounded capacity and the information sent on each link can be arbitrarily delayed. As it allows unbounded transmission delays, this model can accurately abstract a dynamic network with a fixed set of processes.

Based on this model, [1] proposes algorithms to solve the end-to-end communication problem for different criteria. Assuming an upper bound on time a message is being held in the network, they propose the **slide** protocol for solving the end-to-end communication problem. Furthermore they extend the basic slide protocol with a known data dispersal component to optimize the message complexity to $\mathrm{O}(n)$. Finally, they combine an algorithm designed for static network structures (fixed processes and fixed communication channels) with the slide protocol to obtain a protocol that unlike most algorithms for dynamic networks, adequately improves to performance when being run in a static network.

Especially interesting for this thesis is a property called *eventual connectedness*. It means that two nodes are connected at some point during the run (not necessarily direct, but via other nodes). This assumption is also used in different ways in most recent publications about dynamic networks; we will also use it in this thesis to guarantee the termination of our algorithm.

The authors of [6] use a similar model. Again, the set of processes is fixed and the links can dynamically crash or recover. This feature is realized via a finite but unpredictable travel time per link in conjunction with a synchronous model. Each message sent at clock tick p has to arrive before clock tick $p + 1$ on the sending node, and each node knows the state of directly connected links.

The focus of this paper is the exploration of possible combinations of static approaches with dynamic algorithms. Different approaches are assessed with respect to several complexity measurs like communication, space and time complexity.

The main result is a poly-log time, space and communication simulation of synchronous, static protocols on dynamic asynchronous networks. To accomplish this the authors combine the concept of sparse covers to obtain an improved synchronizer protocol for static networks and the concept of local resets to implement a local rollback technique. The result is a protocol that can simulate a static system on a dynamic network.

The main points to gain from these papers, besides the already mentioned results, are that static solutions can be seen as special cases of dynamic ones. Dynamic algorithms often lack runtime, space or message complexity improvements when employed in networks with near-static behavior, however. Moreover, one can conclude that assuming synchrony in a dynamic network is not far fetched, as it can be achieved by some synchronizer algorithm in an asynchronous network.

Round-by-Round models

Rather than considering the possibility of unbounded/infinite delay on the transmission link, [37] focuses on explicit communication failures and the resulting possibility/impossibility borders. The basic model is a synchronous system with dynamic transmission failures. The message sending process is realized via a broadcast from one process to all other processes in the system via point to point communication over a fully connected network.

The authors distinguish 3 types of failures: omission (message is never delivered), corruption (message is delivered with different content), and addition (a message is delivered although no message was sent).

The goal is to achieve agreement among a majority of the processes (i.e., the system consists of n processes with initial value $\in \{0, 1\}$ and at least $k > \frac{n}{2}$ processes shall decide on the same value). They establish conditions on the number of transmission failures that render this problem solveable/unsolvable.

As we are using a similar model with only omission failures, the result showing that consensus is impossible under the assumption of $n - 1$ dynamic omission transmission failures is very important. It gave us a starting point for the research and supported the additional assumptions introduced to make k-set agreement possible.

The work on "round-by-round fault detectors" in [23] has some similarity to the "heard-of" model, which we will discuss subsequently. The main difference is that Gafni focuses on process failures instead of transmission failures. An oracle (a round-by-round fault detector) is used by each process to obtain a set of processes from which it will not be able to receive any data in the current round. Thus, the oracle abstracts away the actual reason for not receiving messages. Moreover, the communication means are also abstracted away, in the sense that it does not matter whether message passing or shared memory is used. Obviously, abstracting away the failure source locally at the receiver via a set of faulty processes produced by the oracle has the benefit that the processes do not have to worry about the reason for and the actual source of the failure. On the down side, dedicated implementations of the oracle have to be provided for every different network model, which may or may not be possible.

The "heard-of" model (abbrivated HO-model) published in [18] is the basis for the model introduced in [10] and hence for the model used in this thesis. The processes are fixed and assumed to be fault-free and their execution progresses as a sequence of lock-step synchronous rounds. It uses a slightly different approach with respect to failures than [23], which also abstracts away the reason why messages are missing (sender crash, receiver failure, etc.). The only information describing communication in each round is a set $H(p, r)$, which consists of all process that have successfully sent a message to p in round r. Rounds are communication-closed, so messages sent in some round r can only be received in the same round r, and are disregarded and hence not included in any $H(p,\dot{})$ if received late. Thus, a run is described by a set $H(p,r)$ for each node, for every round $r \geq 1$, which is determined by the adversary. To make different agreement problems solvable in the HO-model, different predicates restricting the freedom of the adversary are usually required [18]. Note that there is also a Byzantine extension of the HO-model [12]. Obviously, the HO-model is a suitable abstraction for wireless communication, because every participant simply broadcasts its messages but only has knowledge about received information. It is also a reasonable way to hide the different sources of failures in the communication model and handle different communication failures via the HO-sets.

An additional related model is the perception-based model by Biely, Schmid and Weiss ([38]). A sequence of perception matrices, the rows of which are the sets of messages received by some processes, is used to express failures of processes and links. A *hybrid failure model*, which supports different types of process an link failures, is used to restrict the possible perception matrices in a round-by-round fashion. To circumvent the impossibility of transmission failures established in [37], for example, a restriction to the number of transmission failures per process for outgoing and incoming links can be applied. Since these failures are tied to each process on a per round basis, agreement can be achieved in the presence of $O(n^2)$ transmission failures per round.

In [38] the authors discuss a suite of consensus algorithms under this model, ranging from the oral messages (EIG) algorithm to the *Phase Queen* and the *Phase King* algorithms.

The lower bound results established in [38] make it clear that there is no hope to solve consensus without quite strong connectivity guarantees. In particular, without some eventual stablity assumptions, the number of link failures per round has to be bounded for every process.

Consensus and k-set Agreement

The k-set agreement problem is a well known problem in distributed computation. Introduced in [19] it has been studied under many different system models and failure assumptions [2]. Most of these assumptions imply static models, typically involving a foreknown maximum number f of faulty processes and bidirectional communication links between processes.

For synchronous systems, where one assumes full connectivity, algorithms for this problem are rather simple and mostly based on some upper bounds on the required number of rounds, like $\lfloor \frac{f}{k} + 1 \rfloor$. These upper bounds, which for example depend on the number of faulty processes in the network, ensure sufficient information flow between correct processes.

Since k-set agreement is a relaxation of consensus, it has primarily been used to explore the possibility/impossibility border in asynchronous systems. In particular, [36] or [13] establish impossibility results in the *f-resilient, asynchronous* model via arguments from algebraic topology. They prove that for $f \geq k$, where f is the maximum number of crashed processes, k-set agreement is impossible.

As already established in the introduction, such static properties can not be applied for modeling k-set agreement in dynamic systems. A first step towards a more dynamic setting has been made in [9], though. It considers k-set agreement on the basis of the HO-model, where the communication predicate guarantees a stable skeleton graph: Each round is represented by a communication graph G_r, which consists of nodes (processes) and directed edges (p, q) modeling round r communication between process p and g, i.e., when $p \in H(q, r)$. The assumption of a stable skeleton graph guarantees that $\bigcap_{r \geq 1} G^r$ is non- empty, i.e., that there is a non-empty set of edges, which are present in every round.

Moreover, [9] assumes that in each possible set of $k + 1$ nodes of the stable skeleton graph, two nodes share the same parent node in every round. This "two-source" assumption guarantees that the stable skeleton consists of at most k root components i.e., strongly connected component with no incoming edges.

First, an algorithm was proposed that can detect root components via approximating the stable skeleton; since the approximation eventually stabilizes, so do the roots. To solve k-set agreement, it is sufficient to guarantee a common decision among all members of a single root. This is possible, since all members of a root can exchange information (as the root is strongly connected) and can reach all nodes in the associated weakly connected component.

For consensus, a similar but more dynamic model was established by Biely, Robinsion and Schmid in [10]. It is also based on the HO-model: the set of processes is assumed to be fixed and communication is modeled by a sequence of round graphs G_r. Instead of a stable skeleton, however, it is assumed that each round graph is at least weakly connected and has at most one root component. Moreover, eventually, the root component needs to be vertex-stable for a certain number of consecutive rounds to guarantee termination. The model can be placed between class 1 and 3 of the framework established in [17] (note that it is strictly weaker than 3 and stronger than 1).

An algorithm for solving consensus in this model has been proposed which can detect vertex-stable root components and determine their stability intervals. The idea is to optimistically "lock" on $D{+}1$-stable root components (D is an upper bound on the information propagation in a strongly connected component) and to promote the locked on information among all processes in the weakly connected component. If the vertex-stability of a root component holds for further D rounds, the algorithm can safely decide, because it is guaranteed that the locked on information has reached every process in the associated weakly connected component. Interestingly, for a single process to terminate, a much smaller stable root interval is needed than for guaranteeing global termination. Actually the global termination time interval is twice as long $(4D{+}1)$. Thus, it is possible to have some decision in the network without the majority of the processes knowing. We will see that we have a very similar problem for the k-set case, where some processes that already made decisions but are "hidden" in different components could try to promote their own values later, which would violate the agreement part.

Furthermore, [10] shows, via an impossibility result, that in a model with such weak assumptions, consensus is impossible if no root is vertex stable for at least D rounds. Thus, building on D stable intervals, as we are doing in this thesis, seems a reasonable starting point to also solve k-set agreement.

The model in this thesis is the same as used in [10], except that weaker constraints are used to make k-set agreement possible. In particular we will relax the assumption of only one root component per round to at most k.

In [8], the authors establish a framework for k-set agreement *easy impossibility proofs*. It can be use to show impossibility in different models via the same generic theorem, which reduces k-set agreement to consensus in certain executions.

Basically, it uses the concept of *restricted algorithms* for relating executions of the whole system and executions of disconnected components. The relation is based on *indistinguishable runs* for some processes. Essentially, the executions considered assume k disconnected components. Since disconnected components never hear from each other during the whole run, independent decisions can be forced in every component (*partitioning argument*). Obviously, to guarantee the k-agreement property, i.e., no more than k different decisions, this requires to solve consensus within every component. The impossibility of k-set agreement follows from showing that the restricted algorithm cannot solve consensus in at least one component.

In an joint effort with a fellow student (Kyrill Winkler), this approach has been used to derive impossibility results for k-set agreement under natural restrictions of the adversary, which confirm that the constraints under which the presented algorithm operates correctly are very close to the solvability border.

CHAPTER 2

Model

We use the same dynamic network model as in [10], that is, we model a distributed computing system as an ensemble of processes Π. Each process $p_i \in \Pi$ has a unique index i that ranges between 1 and $n = |\Pi|$, the total number of processes in the system. A single process is modeled as a deterministic state machine, which knows its id i but does not necessarily know n. It has an unlimited amount of local memory that comprises its current state and can communicate with other processes by broadcasting messages.

The model is synchronous, i.e., abstracts away time via a sequence of lock step rounds: Round r, $r \geq 1$, comprises the broadcasting of a message, the reception of all these messages at the receiver, and a computing step that processes the received messages (and also determines the message to be send in round $r + 1$). In a synchronous system, all processes take their computation steps simultaneously, and the delay between consecutive steps is big enough such that each message sent in step r is received before step $r + 1$ (r is an element of $\mathbb{N}$ and is also used to enumerate computation steps). A message sent in round r is also labeled with the index r, hence, **round** r is defined by all messages and the single computation step with index r. Conveniently, we use the round numbers as our notion of time.

All communication in the system can hence be interpreted as a sequence of communication graphs G^r, $r \geq 1$, where G^r consists of vertices representing processes and directed edges (p, q) representing the successful transmission of the round r message from p to q. The sequence of G^r is chosen by the adversary, which must adhere to certain restrictions of the "allowed" message loss in order to render k-set agreement solvable.

We will only need two restrictions, namely, **k-majority influence** and **stable interval**. The second one has to hold eventually once during an execution, whereas the first one has to hold perpetually until the second one holds. We will discuss these restrictions (later on called model properties) in more detail in Section 2.5. But first we will give a formal definition of the model in the next subsections.

2.1 Failure assumption

The failure assumptions in this model are entirely focused on link failures between processes. If a message is sent by p in round r and not received by q in r, there is no edge (p, q) in G^r even if the message would arrive later. Whereas the model cannot handle value faulty messages, it also captures crashed processes.

2.2 Computation

Computation in our system is structured in a sequence of communication-closed rounds, where processes (conceptually) advance via an infinite number of rounds in lock-step. In every round r, $r > 0$, processes can broadcast a round r message of arbitrary content and perform some local computation based on the received round r messages and their current (local) state. As the computational model is equivalent to [10], we will take the model definitions from [10]: For every $p \in \Pi$ and each round $r > 0$, let $S_p^r \in S_p$ be the state of p at the beginning of round r; the initial state is denoted by $S_p^1 \in S_p^1 \subset S_p$. The round r computation of process p is determined by the following two functions that make up the algorithm of p: The message sending function $M_p : S_p \to M$ determines the message m_r broadcast by p in round r, based on p's state S_p^r at the beginning of round r. We assume that some (possibly empty) message is broadcast in every round, to all (current!) neighbors of p. The transition function $T_p : S_p \times 2^{(\Pi \times M)} \to S_p$ takes p's state S_p^r at the beginning of round r and a set μ_r of pairs of process ids and messages, which uses it to represent the round r messages received by p from other processes in the system and compute the successor state S_p^{r+1}. We assume that, for each process q, there is at most one $(q, m_q^r) \in \mu_p$ such that m_q^r is the message q sent in round r. Note that neither M_p nor T_p need to involve n, i.e., the algorithms executed by the processes may be uniform w.r.t. the network size n: Which processes a process actually receives from in round r depends solely on the underlying communication graph of round r, which we define in section 2.3.

2.3 Communication

Again we use the same model as in [10], resulting in the following definition of our communication structure: The network topology is modeled as an infinite sequence of simple directed graphs $G^1, G^2, \ldots$, which is fixed by an adversary having access to the process states. For each round r, we denote the round r communication graph by $G^r = (V, E^r)$, where each node of the set V is associated with one process from the set of processes Π, and where E^r is the set of directed edges for round r, such that there is an edge from p to q, denoted as $(p \to q)$, iff q receives p's round r message (in round r). Figure 2.1 shows a possible sequence of graphs for a network of 5 processes, for rounds 1 to 4. For any graph G, we will use the notation $V(G)$ and $E(G)$ to refer to the set of vertices respectively edges of G, i.e., it always holds that $G = (V(G), E(G))$. For deterministic algorithms, as considered in this thesis, a run is completely determined by the input values assigned to the processes and the sequence of communication graphs. Therefore, the fact that the adversary knows only the initial values does not pose a limitation to its power. To simplify the presentation, we will denote a process and the associated node in the communication graph by the same symbols and omit the set from which it is taken if there is no ambiguity. We will henceforth write $p \in G^r$ and $(p \to q) \in G^r$ instead of $p \in V(G)$ resp. $(p \to q) \in E^r$. The neighborhood of p in round r is the set of processes N_p^r that p receives messages from in round r, formally, $N_p^r = \{q | (q \to p) \in G\}$.

Configurations

Based on the local states S_p^r we define the a configuration as follows:

Definition 1 (round r configuration)**.** *A round r configuration $\mathcal{C}^r$ is the vector of states S_p^r for all $p \in \Pi$.*

An **execution** or **run** of an algorithm is an infinite alternating sequence of configurations $\mathcal{C}^i$ and graphs G^i starting with the initial configuration $\mathcal{C}^1$. $\mathcal{C}^{r+1}$ is reached from $\mathcal{C}^r$, by determining, for each p in parallel, $\vec{\mu}_p^r$ based on p's incoming edges in G^r and applying T_p to p's state in $\mathcal{C}^r$ and $\vec{\mu}_p^r$.

2.4 Network properties

Similarly to the classic notion of "happened-before" [30], we say that a process p (causally) influences process q in round r, expressed by $(p \overset{r}{\rightsquigarrow} q)$ or just $(p \rightsquigarrow q)$ if r is clear from the context, iff either (i) $p \in N_q^r$, i.e., if q has an incoming edge $(p \to q)$ from p in G^r , or (ii) if $q = p$, i.e., we assume that p always influences itself in a round. We say that there is a (causal) chain of length $k \geq 1$ starting from p in round r to q, graphically denoted by $(p \overset{r[k]}{\rightsquigarrow} q)$ or simply $(p \rightsquigarrow q)$, if there exists a sequence of not necessarily distinct processes $p = p_0, ..., p_k = q$ such that p_i influences p_{i+1} in round $r + i$, for all $0 \leq i < k$. If k is irrelevant, we just write $(p \overset{r}{\rightsquigarrow} q)$ or just $(p \rightsquigarrow q)$ and say that p (in round r) causally influences q.

Causal distance

The **causal distance** $cd^r(p, q)$ at round r from process p to process q is the length of the shortest causal chain starting in p in round r and ending in q, formally, $cd^r(p, q) := min\{k | (p \overset{r[k]}{\rightsquigarrow} q)\}$. Note that we assume $cd^r(p, p) = 1$. The following Lemma 1 shows that the causal distance in successive rounds cannot arbitrarily decrease.

Lemma 1 (Causal distance in successive rounds [10, Lemma 1])**.** *For every round $r \geq 1$ and every two processes $p, q \in \Pi$, it holds that $cd^{r+1}(p, q) \geq cd^r(p, q) - 1$. As a consequence, if $cd^r(p, q) = \infty$, then also $cd^{r+1}(p, q) = \infty$.*

Proof. Since $(p \rightsquigarrow p)$ in every round r, the definition of causal distance trivially implies $cd^r(p, q) \leq cd^{r+1}(p, q) + 1$. □

Note that the causal distance in directed graphs is not necessarily symmetric. Moreover, if the adversary chooses the graphs G^r such that not all nodes are strongly connected, the causal distance can even be infinite. In fact, even if G^r is strongly connected for round r (but not for rounds $r' > r$), $cd^r(p, q)$ can be infinite.

Strongly connected components

As we will not consider the whole communication graph to be strongly connected in this paper, we make use of the notation of **strongly connected components** (SCC). We write C_p^r to denote the (unique) SCC of G^r that contains process p in round r or simply C^r if p is irrelevant. It is apparent that $cd^r(p,q)$ and $cd^r(q,p)$ may be infinite even if $q \in C_p^r$. In order to be able to argue (meaningfully) about the maximal length of causal chains within an SCC, we also need some "continuity property" over rounds. This leads us to the crucial concept of a **I-vertex-stable strongly connected component**, denoted as C^I : It requires that the set of vertices of a strongly connected component C remains stable throughout all rounds in the nonempty interval $I = [r,s]$, $s \geq r$. Note carefully that we assume $|I| = s - r + 1$, since I ranges from the beginning of round r to the *end* of round s; hence, $[r,r]$ is not empty but rather represents round r. Its topology may undergo changes, but must form an SCC in every round. Formally, C^I being vertex-stable during I requires that $\forall p \in C^I, \forall x \in I : V(C_p^x) = V(C^I)$, where $V(C)$ denotes the set of processes in the SCC C. The important property of C^I is that information is guaranteed to spread to all vertices of C^I if the interval I is large enough (cf. Lemma 3).

Let the round r **causal diameter** $\phi^r(C^I)$ of a vertex-stable SCC C^I be the largest causal distance $cd^r(p,q)$ for any $p,q \in C^I$. The causal diameter $\phi(C^I)$ of a vertex-stable SCC C^I in I is the largest causal distance $cd^x(p,q)$ starting at any round $x \in I$ that "ends" in I, i.e., $x + cd^x(p,q) - 1 \in I$. If there is no such causal distance (because I is too short), $\phi(C^I)$ is assumed to be infinite. Formally, for $I = [r,s]$ with $s \geq r$,

$$\phi(C^I) = min\{max\{\phi^x(C^I) | x \in [r,s] \text{ and } x + \phi^x(C^I) - 1 \leq s\} \cup \infty\}$$

.

If C^I consists only of one process, then we obviously have $\phi(C^I) = 1$. The following Lemma 2 establishes a bound for $\phi(C^I)$ also for the general case.

Lemma 2 (Bound on causal diameter [10, Lemma 2]). *Given some $I = [r,s]$ and a vertex-stable SCC C^I with $|C^I| \geq 2$: If $s \geq r + |C^I| - 2$, then $\phi(C^I) \leq |C^I| - 1$.*

Proof. Fix some process $p \in C^I$ and some r' where $r \leq r' \leq s - |C^I| + 2$. Let $P_0 = \{p\}$, and define for each $i > 0$ the set $P_i = P_{i-1} \cup \{q : \exists q' \in P_{i-1} : q' \in N_q^{r'+i-1}\}$. P_i is hence the set of processes q such that $(p \overset{r'[i]}{\rightsquigarrow} q)$ holds. Using induction, we will show that $|P_k| \geq min\{|C^I|, k+1\}$ for $k \geq 0$. Induction start $k = 0 : |P_0| \geq min\{|C^I|, 1\} = 1$ follows immediately from $P_0 = \{p\}$. Induction step $k \to k+1, k \geq 0$: First assume that already $|P_k| = |C^I| \geq min\{|C^I|, k+1\}$; since $|P_{k+1}| \geq |P_k| = |C^I| \geq min\{|C^I|, k+1\}$, we are done. Otherwise, consider round $r' + k$ and $|P_k| < |C^I|$: It follows from strong connectivity of $G^{r'+k} \cap C^I$ that there is a set of edges from processes in P_k to some non-empty set $L_k \subseteq C^I \setminus P_k$. Hence, we have $P_{k+1} = P_k \cup L_k$, which implies $|P_{k+1}| \geq |P_k| + 1 \geq k + 1 + 1 = k + 2 = min\{|C^I|, k+2\}$ by the induction hypothesis. Thus, in order to guarantee $C^I = P_k$ and thus $|C^I| = |P_k|$, choosing k such that $|C^I| = 1 + k$ and $k \leq s - r' + 1$ is sufficient. Since $s \geq r' + |C^I| - 2$, both conditions can be fulfilled by choosing $k = |C^I| - 1$. Moreover, due to the definition of P_k , it follows that $cd^r(p,q) \leq |C^I| - 1$ for all $q \in C^I$. Since this holds for any

p and any $r' \leq s - |C^I| + 2$, we finally obtain $|C^I| - 1 \geq \phi^r(C^I)$ and hence $|C^I| - 1 \geq \phi(C^I)$, which completes the proof of Lemma 2. □

Given this result, it is tempting to assume that, for any vertex-stable SCC C^I with finite causal diameter $\phi(C^I)$, any information propagation that starts at least $\phi(C^I) - 1$ rounds before the final round, will reach all processes in C^I within I. This is not generally true, however, as the following example for $I = [1, 3]$ and a vertex-stable SCC of four processes shows: If G^1 is the complete graph whereas $G^2 = G^3$ is a ring, $\phi(C^I) = 1$, but information propagation starting at round 2 does not finish by the end of round 3. However, the following Lemma 3 gives a bound on the earliest starting round that guarantees this property.

Lemma 3 (Information propagation [10, Lemma 3])**.** *Suppose that C^I is an I-vertex-stable strongly connected component of size ≥ 2 that has $\phi(C^I) < \infty$, for $I = [r, s]$, and let x be the maximal round where $x + \phi^x(C^I) - 1 \leq s$. Then,*
(i) for every $x' \in [r, x]$, it holds that $x' + \phi^{x'}(C^I) - 1 \leq s$ and $\phi^{x'}(C^I) \leq \phi(C^I)$ as well, and
(ii) $x \geq max\{s - |C^I| + 2, r\}$.

Proof. Since $\phi(C^I) < \infty$, the maximal round x always exists. Lemma 1 reveals that for all $p, q \in C^I$, we have $x - 1 + cd^{x-1}(p, q) - 1 \leq x + cd^x(p, q) - 1 \leq s$, which implies $x' + cd^{x'}(p, q) - 1 \leq s$ for every x' where $r \leq x' \leq x$ and proves (i). The bound given in (ii) follows immediately from Lemma 2. □

Since we will frequently require a vertex-stable SCC C^I that guarantees bounded information propagation also for "late" starting rounds and we also want to cover scenarios where the adversary guarantees better-than-worst-case information propagation in C^I, we introduce the following Definition 2.

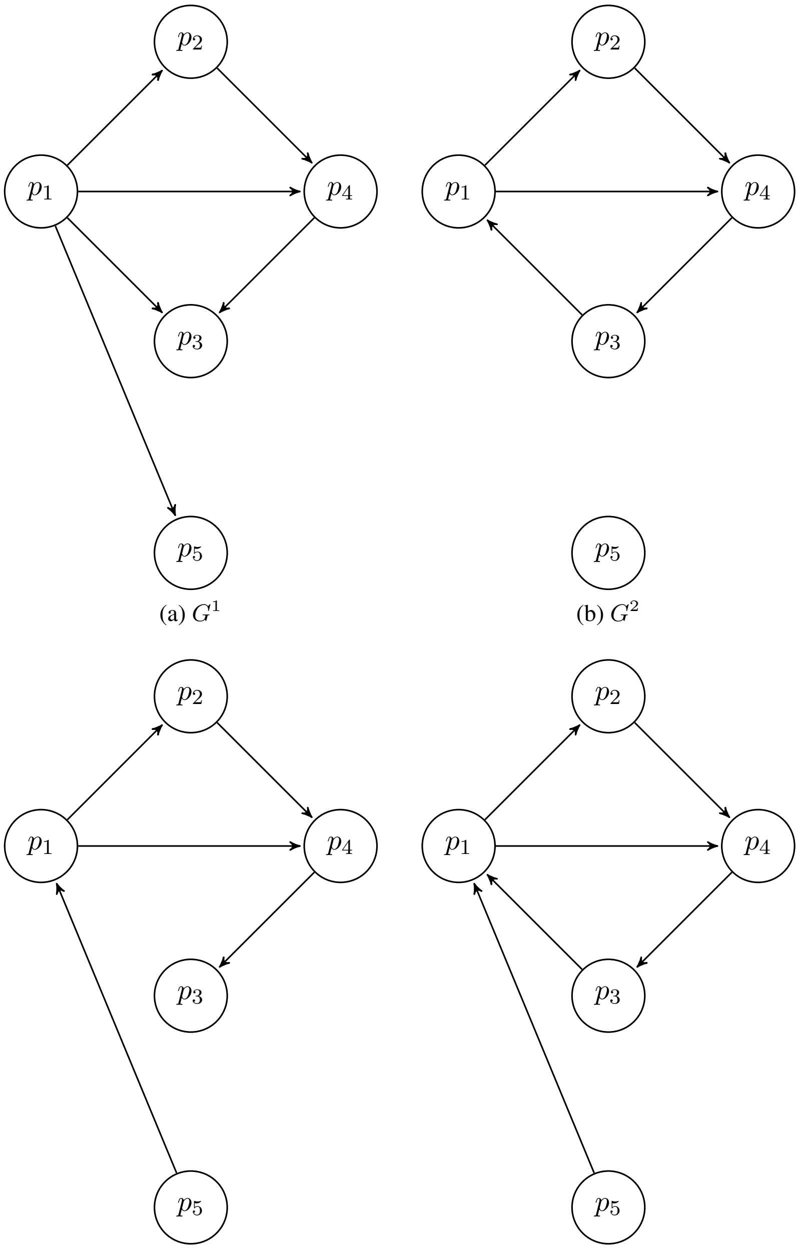
Definition 2 (D-bounded I-vertex-stable SCC [10, Definition 1])**.** *An I-vertex-stable SCC C^I with $I = [r, s], s \geq r$, is D-bounded for $D \leq |I|$, iff $\forall x \in [r, s - D + 1] : \phi^x(C^I) \leq D$.*

Note that not every vertex-stable SCC C^I with $|I| \geq D$ needs to be D-bounded, for some given D. However, we can of course restrict the adversary to obey this requirement. Moreover, due to Lemma 2, every vertex-stable SCC C^I with $|I| \geq D$ is D-bounded for $D \geq |C^I| - 1$, which in turn means $|C^I| - 1$ is an upper bound for D. Note that such vertex-stable SCCs are necessarily $n - 1$-bounded.

2.5 Required connectivity properties

The model above gives us a convenient way to describe the communication between processes, but does not yet restrict communication in any way. Without restrictions of the adversary, however, the communication graph may perpetually be partitioned into $k + 1$ (weakly connected) components, making k-set agreement trivially impossible [9].

Thus, we need some kind of restriction on the number of components that cannot exchange information between each other during the execution.



(a) G^1

(b) G^2

(c) G^3

(d) G^4

Figure 2.1: Example of a 4 round execution G^1, G^2, G^3, G^4

For example, we could require that all communication graphs are split into at most k perpetually stable isolated SCC's, but this seems overly restrictive. For example two SCCs C and C' might be connected (uni-directionally) such that processes in C' can observe the computation in C but not vice versa. **Root components**, which have already been introduced in the context of [10], offer a less restrictive way to specify partitioning in a distributed system.

Root Components

A root component is a strongly connected component that has no incomming edges from outside the component. Again, it is immediate, that if we have $k + 1$ perpetual root components in an execution, k-set agreement is not possible. Thus, one has to restrict the number of root components per round to at most k. It will turn out in Chapter 4, however, that additional properties will be required in order to make k-set agreement solvable.

But at first we will prove a few facts about root components and the influence of root components on the whole graph structure. Each round is defined by G^r, which may also consist of several disconnected subparts $G_1^r(V_1^r, E_1^r), G_2^r(V_2^r, E_2^r), ...$, see Figure 2.1 (b) for example. Obviously, every subgraph G_i^r is at least weakly connected.

Lemma 4 (Minimum amount of roots)**.** *Each graph G_i has at least one root component.*

Proof. From basic graph theory we know that each weakly connected graph can be abstracted by a condensed graph (or component graph) G_{SCC} by contracting each strongly connected component of G_i to a single vertex. Moreover, this graph is by definition weakly connected and does not contain any strongly connected components, i.e., is acyclic. If G_{SCC} contains a node with no incoming edge, which corresponds to a root component by definition, we are done.

Otherwise, pick a random vertex in G_{SCC}. We follow the incoming edge (there has to exist one, otherwise we already found a root contradicting the assumption) of this node to the next node, and repeat this step $|V(G_{SCC})| - 1$ times. Because G_{SCC} is an acyclic graph with at most n nodes (every node in G_i could form a strongly connected component itself), we have visited every node and hence found a cycle. Since G_{SCC} is acyclic, this provides the required contradiction.

□

From Lemma 4, we immediately obtain Corollary 1:

Corollary 1 (Relation between roots and weakly connected components)**.** *If G has at most k root components, then it has at most k weakly connected components.*

Proof. Assume that $k + 1$ components exist. We apply Lemma 4 to each subgraph defined by one of the $k + 1$ weakly connected components and obtain thereby $k + 1$ root components in G. A contradiction. □

Since every root component is also an SCC, all the related definitions and results can be carried over:

We define a I-**vertex-stable root component** R^I (abbreviated I-**VSRC**) as an I-vertex-stable strongly connected component C^I that is a root component in every $r \in I$. Obviously, Lemmas 2 and 3 also apply to R^I.

Definition 3 (D-bounded I-vertex-stable root [10, Definition 2]). *An I-vertex-stable root component R^I with $I = [r, s], s \geq r$, is D-bounded, for $D \leq |I|$, iff $\forall x \in [r, s-D-1] : \phi^x(R^I) \leq D$.*

Our next step is to formalize the concept of information propagation from root components to the rest of the network. This is done by means of a generalization of the network causal diameter introduced for consensus in [10]. Since it will not matter which root component actually influences some process q outside any root component, the **network causal diameter** E is defined as follows:

Definition 4 (E-bounded network). *A network is E-bounded, iff, in every run, there exists a set S of $\ell \geq 1$ D-bounded vertex-stable root components $R_1^{[r_1,s]}, \dots, R_\ell^{[r_\ell,s]}$ where $\forall r' \in \{r_1, \dots r_\ell\}$: $s - r' \geq 3D + E$ and $\forall q \in \Pi : \exists R_x^I \in S : \exists p \in R_x^I : cd^{s-E+1}(p, q) \leq E$.*

The following Lemma 5 proves that the network causal diameter E is bounded by $n-1$, provided there exist $\ell \geq 1$ I-vertex-stable root components $R_1^I, \dots, R_\ell^I$ with $|I| \geq n-1$, simultaneously.

Lemma 5 (Bound on network causal diameter). *Suppose there is some interval $I = [r, s]$ where ther $\ell \geq 1$ I-vertex-stable root components $R_1^I, \dots, R_\ell^I$. If $s \geq r + n - 2$ and $n \geq 2$, then $E \leq n - 1$.*

Proof. Fix arbitrary processes $p_1 \in R_1^I, \dots, p_\ell \in R_\ell^I$ and some r' where $r \leq r' \leq s - n + 2$. Let $P_0 = \{p_1, \dots, p_\ell\}$, and define for each $i > 0$ the set $P_i = P_{i-1} \cup \{q : \exists q' \in P_{i-1} : q' \in N_q^{r'+i-1}\}$. P_i is hence the set of processes q such that $(p \overset{r'[i]}{\rightsquigarrow} q)$ holds for at least one $p \in P_0$. Using induction, we will show that $|P_k| \geq min\{n, k+1\}$ for $k \geq 0$. Induction start $k = 0 : |P_0| \geq min\{n, 1\} = 1$ follows immediately from $P_0 = \{p_1, \dots, p_\ell\}$ with $\ell \geq 1$. Induction step $k \to k+1, k \geq 0$: First assume that already $|P_k| = n \geq min\{n, k+1\}$; since $|P_{k+1}| \geq |P_k| = n \geq min\{n, k+1\}$, we are done. Otherwise, consider round $r' + k$ and $|P_k| < n$: Since every node $q \in \Pi$ is in a weakly connected component containing at least one root in every round, hence also in $G^{r'+k}$, there is a set of edges from processes in P_k to some non-empty set $L_k \subseteq \Pi \setminus P_k$. Hence, we have $P_{k+1} = P_k \cup L_k$, which implies $|P_{k+1}| \geq |P_k| + 1 \geq k + 1 + 1 = k + 2 = min\{n, k+2\}$ by the induction hypothesis. Thus, in order to guarantee $\Pi = P_k$ and thus $n = |P_k|$, choosing k such that $n = 1 + k$ and $k \leq s - r' + 1$ is sufficient. Since $s \geq r' + n - 2$, both conditions can be fulfilled by choosing $k = n - 1$. Moreover, due to the definition of P_k, it follows that for all $q \in \Pi$ there is some $p \in P_0$ with $cd^r(p, q) \leq n-1$. Since this holds for any choice $p_1 \in R_1^I, \dots, p_\ell \in R_\ell^I$ for P_0 and any $r' \leq s - n + 2$, we can conclude $n - 1 \geq E$, which completes the proof of Lemma 5. □

The combination of Lemma 5 and Lemma 2 in conjunction with Definition 4 shows that the causal diameter D and the network causal diameter E are both $n-1$ bounded, provided a set of I-vertex stable roots with $|I| \geq 4n - 4$ exist.

Connectivity properties

In this section, we provide two conditions (Assumption 1 and Assumption 2), the conjunction of which will be sufficient for making k-set agreement solvable. Note that choosing these constraints has been guided by avoiding the impossibility results obtained by Theorems 4 and 5 in Chapter 4, which exploit different weaknesses of the assumption of at most k root components per round (Assumption 4) only. Although these additional constraints may appear somewhat contrived at a first glance, we argue below that they are implied by some very natural stronger assumptions, for which the k-set algorithm described in Chapter 3 hence also works correctly.

To avoid non-terminating (i.e. forever undecided) executions, we add the following *stable interval* constraint:

Assumption 1 (Stable interval). *In every run, all vertex-stable root components R^I with $|I| \geq D+1$ are D-bounded, for some $D > 0$. Moreover, the network is E-bounded for some $E > 0$.*

Note that the second part of Assumption 1 is implied by the stronger condition $\forall r \geq r_{GST}\colon \mathcal{G}^r = \mathcal{G}^{r+1}$, which asserts an eventually stable communication graph with $\ell \leq k$ identical vertex-stable root components.

Next, in order to also circumvent the executions violating k-agreement constructed in Theorem 5, we introduce the *majority influence* constraint (cf. Assumption 2). Informally, it guarantees some (minimal) information flow between vertex-stable root components that exist at different times. More specifically, it ensures that not *all* VSRCs that are stable long enough such that some processes could have decided can become isolated later on.

Given some run ρ, we denote by $\mathbb{V}_d$ the set of all root components which are vertex stable for at least d subsequent rounds in ρ.

Let R_{cur} be vertex stable in $[r_{cur}, s_{cur}]$ and R_{suc} be vertex stable in $[r_{suc}, s_{suc}]$ with $r_{suc} > s_{cur}$.

Definition 5 (Influence). *We say that some process $a \in R_{cur}$ **influences** some process $b \in R_{suc}$ and write $a \hookrightarrow b$ with $\hookrightarrow \subseteq \Pi^2$ iff there exists a process $a \in R_{cur}$ and a process $b \in R_{suc}$ s.t. the causal distance $cd^{s_{cur}+1}(a,b) \leq r_{suc} - s_{cur}$.*

The following Definition 6 specifies the set of processes in R_{suc} which are influenced by some process in R_{cur}.

Definition 6 (Influenced Set). *$IS(R_{cur}, R_{suc}) := \{b \in R_{suc} \mid \exists a \in R_{cur}\colon a \hookrightarrow b\}$*

The *majority influence* between R_{cur} and R_{suc} guarantees that R_{cur} influences a set of processes in R_{suc}, which is greater than any influenced set of a root that did not influence R_{cur}

Definition 7 (Majority influence). *We say that a VSRC $R_{cur} \in \mathbb{V}_{2D+1}$ exercises a **majority influence** on a VSRC $R_{suc} \in \mathbb{V}_{2D+1}$, denoted $R_{cur} \hookrightarrow_m R_{suc}$ with $\hookrightarrow_m \subseteq \mathbb{V}^2_{2D+1}$, iff*
$\forall R \in \mathbb{V}_{D+1}$ with $IS(R, R_{cur}) = \emptyset\colon$ $IS(R_{cur}, R_{suc}) > IS(R, R_{suc})$ and
$\forall R \in \mathbb{V}_{D+1}$ with $IS(R, R_{cur}) \neq \emptyset\colon$ $IS(R_{cur}, R_{suc}) \geq IS(R, R_{suc})$.

The following Assumption 2 guarantees that among the set of all $(2D+1)$-VSRCs, i.e. roots that possibly lead to a decision, at most k exist, which are not majority influenced by any other $(2D+1)$-VSRC, see Figure 2.2.

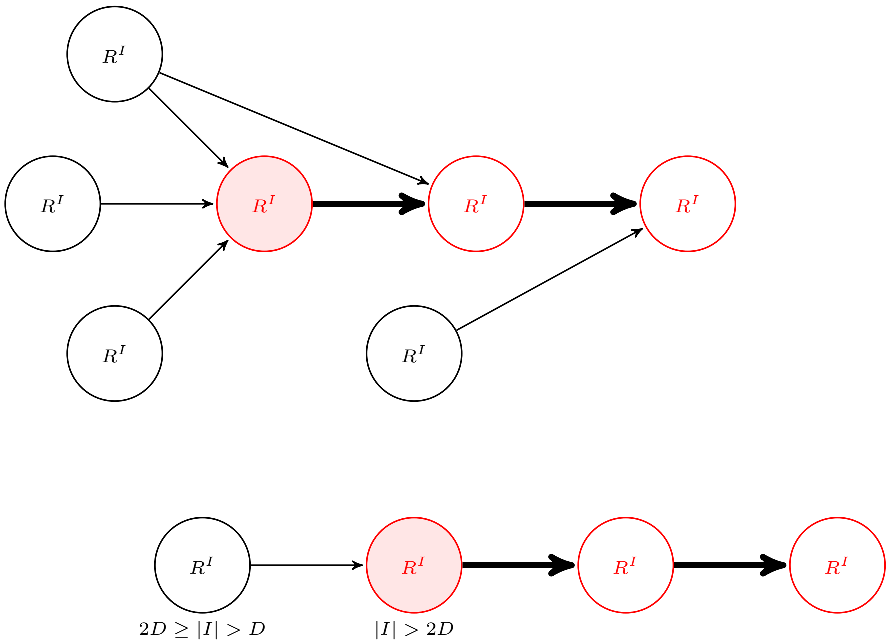


Figure 2.2: Stable root components influencing other root components, for $k = 2$. A bold arrow indicates **majority influence**. The figure also shows a temporal order between the roots, meaning the farther left roots precede the roots on the right. Red nodes have an interval length greater $2D$, black nodes have an interval length between $2D$ and D.

Assumption 2 (k-majority influence)**.** *With the exception of k elements of $\mathbb{V}_{2D+1}$, it has to hold that $\forall v \in \mathbb{V}_{2D+1} \exists v' \in \mathbb{V}_{2D+1} : v' \hookrightarrow_m v$*

In Chapter 3, we will prove that the conjunction of Assumption 1 and Assumption 2 is sufficient for solving k-set agreement, by providing an algorithm and proving its correctness.

2.6 k-set agreement

In this section, we will provide detailed definitions of the **k-set agreement** problem and related concepts like **k-uniformity**, **consensus** and **valence of configurations**.

As we have given an informal description of the problem in the first chapter only, we now define k-set agreement properly.

We assume that the local state $S^r_{p_i}[x]$ of p_i (at the beginning of round r) contains a variable x^r_i, with $x_i = x^1_i \in S^1_{p_i}$ defining its initial value, and a (write-only) variable y_i, with $y^1_i = \perp$ initially undefined. All x_i are chosen from a finite set V with $|V| \geq n$. To solve k-set agreement, each p_i has to eventually assigne a value to y_i exactly once, such that the following three properties (summarized in Definition 8 below) are satisfied:

Validity This property states that every value y_i picked by p_i as its decision value has to be one of the values x with which at least one process p_j started. This prevents trivial solutions, like the one where every process always sets y_i to 0 and terminates.
Validity is normally a simple property to fulfill. Many algorithms only add initial values to the set of candidate decision values for y_i and use a function that assigns the minimum or the maximum value to y_i.

k-Agreement This property is the core of the whole problem. It ensures that every non faulty process that assigns a value to y_i in some round does it in a manner, such that an omniscient observer never finds more than k different assigned values system wide.

Termination Every correct process has to irrevocably decide on a value y_i in some round. This implies that there is a round r where all processes have assigned a non-$\perp$ value to y_i.

Even if it looks quite simple and easy to implement, termination proved to be very difficult to gurantee in dynamic networks.

Definition 8 (k-set agreement.)**.** *For all processes $p_i \in \Pi$, it has to hold that*

- *Validity:* $\forall r \geq 1, \forall p_i \in \Pi : y_i^r \neq \perp \rightarrow y_i \in \{x_0, x_1, ...x_{n-1}\}$
- *k-Agreement:* $\forall r \geq 1, \forall p_i \in \Pi : |\{y_i^r \neq \perp | 0 \leq i \leq n-1\}| \leq k$
- *Termination:* $\forall p_i \in \Pi : (\exists r \geq 1 : y_i^r \neq \perp) \wedge (\forall r \geq 1 : y_i^r \neq \perp \Rightarrow y_i^r = y_i^{r+1})$.

k-uniformity and adaptivity

Since network connectivity and failures are under the control of the adversary, the quality of the results of an algorithm is strongly connected to the behavior of the adversary. Normally, an algorithm is designed such that it can always deliver a certain quality of the results, even in a worst-case scenario. To accomplish this, some kind of knowledge regarding worst-case scenarios, like the maximum number of failures f, is encoded in the algorithm. Typically k-set agreement algorithms also use k somewhere in the code.

However, algorithms can sometimes be designed in a way that they additionally can improve the results if the adversary provides an average or best case scenario. This design property is called adaptivity. An algorithm that provides adaptivity must be able to handle failures or destructive behavior during runtime and provide results according to the actual scenario.

The algorithm introduced in this thesis is k-uniform, which implies adaptivity: k-uniform means that the algorithm is not aware of k. Thus the quality of the result (i.e., the number of different decisions) is entirely dependent on the adversary, i.e., the actual network connectivity in a run. Assumption 1 and Assumption 2 guarantee a worst case result of k different decision values, for any particular value of k. Hence, in a run where $k = 5$, the algorithm in Figure 2.2 will provide at most 5 different decision. If the adversary provides a network that adheres to Assumption 1 and Assumption 2 for $k = 1$ in a run, then the algorithm will actually solve consensus in this run.

Consensus

Consensus is a special case of k-set agreement, where k is set to 1. This implies that the size of the set of values on which each process can decide is reduced to 1, thus every process has to decide on the same value. This special case is interesting for two reasons.

First, it is the optimum regarding system-wide agreement, and hence preferable from an application perspective.

Second, consensus is the core element of two papers ([7],[10]), which are the basis of this Master thesis. In [10], consensus is solved under a very similar model of in dynamic networks. In [7], the ability to solve consensus (or, to be more precise, not to solve consensus) is used to derive short and easy impossibility results for k-set agreement under different models. In fact, solving consensus in k isolated components also solves k-set agreement.

Valence of configurations

We have already established the definition of a configuration in Definition 1. The valence of a configuration is a powerful tool for deriving impossibility results for consensus [22]. We also will use this concept in the last chapter of this thesis.

A configuration $\mathcal{C}^r$ of a binary consensus algorithm ($v = \{0, 1\}$) is called

- *v-decided*, if all processes have $y_i^r = v$.
- *v-valent*, if for all configurations reachable from $\mathcal{C}^r$ that are w-decided it holds that $w = v$.
- *univalent*, if there is a v such that $\mathcal{C}^r$ is v-valent.
- *bivalent*, if it is not univalent.

CHAPTER 3

Algorithm

In this section, we provide a k-set agreement algorithm for the model in Chapter 2 and prove its correctness. The algorithm consists of two parts:

(i) A network approximation algorithm, which provides every process with *local* estimates of the communication graphs G^r of all past rounds. It provides a routine called inStableRoot(I), which uses the local network estimates to determine a posteriori whether the local process was member of an I-vertex-stable root component.

(ii) The k-set agreement core algorithm, which uses inStableRoot() to find out whether the local process was part of a $D+1$-VSRC R. If so, all members are guaranteed to have received the input values from each other. The algorithm hence considers this event as a possible candidate for a decision and *locks* it. If R finally turns out to have extended to a $2D+1$-VSRC, the algorithm decides on the maximum of all known values and broadcasts its decision in the following rounds.

To simplify the presentation, we will state the two algorithms independently of each other. We assume that the complete round r computing step of the network approximation algorithm is executed before the round r computing step of the k-set algorithm, and that the round r message of the former is piggybacked on the round r message of the latter (and hence sent simultaneously). Note carefully that this implies that the round r computing step of the k-set algorithm can already access the *result* of the round r computation of the network approximation algorithm, i.e., the state at the beginning of round $r+1$.

3.1 The local network approximation algorithm

At the beginning, each process has no information about the graph, and knows its own initial value only. To successfully solve the problem at hand, it needs to know the values in the network structure around it. To achieve this, it employs an algorithm that can approximate the network based on the gathered information accurately enough. Algorithm 1 allows each process to build

Algorithm 1 *Local network approximation*

```
Provides callable routine inStableRoot().

Variables and Initialization:
$A_p := \langle V_p, E_p \rangle$ initially $(\{p\}, \emptyset)$ {weighted digraph without multi-edges and loops}

send round $r$ message:
send $\langle A_p \rangle$ to all current neighbors

receive round $r$ message:
save all neighbor messages of round $r$

Round $r$: computation:
for $q \in \mathcal{N}_p^r$ and $q$ sent message $\langle A_q \rangle$ in $r$ do
    if $\exists$ edge $e = (q \xrightarrow{T} p) \in E_p$ then
        replace $e$ with $(q \xrightarrow{T'} p)$ in $E_p$ where $T' \leftarrow T \cup \{r\}$
    else
        add $e := (q \xrightarrow{\{r\}} p)$ to $E_p$
    $V_p \leftarrow V_p \cup V_q$

for every pair of nodes $(p_i, p_j) \in V_p \times V_p, p_i \neq p_j$ do
    if $T' = \bigcup\{S \mid \exists q \in \mathcal{N}_p^r : (p_i \xrightarrow{S} p_j) \in E_q\} \neq \emptyset$ then
        replace $(p_i \xrightarrow{T} p_j)$ in $E_p$ with $(p_i \xrightarrow{T \cup T'} p_j)$; add $(p_i \xrightarrow{T'} p_j)$ if no such edge exists


predicate Callable routine inStableRoot($I$)
    Let $A_p|s = (V_p^s, \{(p_j \xrightarrow{T} p_i) \in E_p \mid s \in T\})$
    Let $C_p|s$ be $A_p|s$ if it is strongly connected, or the empty graph otherwise.
    return $C_p$ if for all $s_1, s_2 \in I : C_p := V(C_p|s_1) = V(C_p|s_2) \neq \emptyset$, otherwise return $\emptyset$
```

an abstraction in form of a TVG (which fits the model of Section 2.3) of the dynamic network. Its set of edges consists of entries of the form $(p_i \xrightarrow{T} p_j)$, where T is the set of all rounds when p_j has received a message from p_i.

Initially A_p consist of $(\{p\}, \{\})$ only. At the beginning of each round, each process q broadcasts A_q to all its neighbors. Thus, p receives a set $\{A_{q_i} | q \in N_p^r\}$ in round r. Based on this information, p updates A_p and adds new nodes and edges accordingly. This update cycle repeats itself each round.

The routine **inStableRoot** returns the members R of the root component iff p is part of the root for the given interval, or else $\emptyset$. It is implemented by checking the "projection" $A_p|r$ of A_p for round r, which is the graph induced by all edges $(p_i \rightarrow p_j)$, where $(p_i \xrightarrow{T} p_j)$ with $r \in R$ is in A_p, is s.c. (strongly connected)).

The algorithm does not guarantee that the real sequence of communication graphs $G^1, G^2, ...$ is perfectly abstracted by A_p, more precisely $A_p|s = G^s$ cannot be guaranteed. But it will turn out that the approximation is accurately enough to solve our problems. In the following section, we will prove that **inStableRoot** correctly returns R if p is part of the root R.

Proof of Correctness for Algorithm 1

For the rest of this chapter we assume that Assumption 1 and Assumption 2 hold. We start out with a proof showing a correct under approximation of G^t for every t.

Lemma 6 ([11, Lem. 7]). *If $A_p|t$ contains $(v \to w)$ at the end of round r, then*

(i) $(v \to w) \in G^t$, *i.e.,* $A_p|t \subseteq G^t$,

(ii) $A_p|t$ *also contains* $(v' \to w)$ *for every* $v' \in N_w^t \subseteq G^t$.

Proof. We first consider the case where $r < t$, then at the end of round r $A_p|t$ is empty, i.e., there are no edges in $A_p|t$. As the precondition of the Lemma's statement is false, the statement is true.

For the case where $r \geq t$, we proceed by induction on r:

Induction base $r = t$: If $A_p|t$ contains $(v \to w)$ at the end of round $r = t$, it follows from $A_q|t = \langle \{q\}, \emptyset \rangle$ at the end of every round $r < t$, for every $q \in \Pi$, that $w = p$, since p is the only processor that can have added this edge to its graph approximation. Clearly, it did so only when $v \in N_p^t$, i.e., $(v \to w) \in G^t$, and included also $(v' \to w)$ for every $v' \in N_p^t$ on that occasion. This confirms (i) and (ii).

Induction step $r \to r+1$, $r \geq t$: Assume, as our induction hypothesis, that (i) and (ii) hold for any $A_q|t$ at the end of round r, in particular, for every $q \in N_p^{r+1}$. If indeed $(v \to w)$ in $A_p|t$ at the end of round $r+1$, it must be contained in the union of round r approximations

$$U = (A_p|t) \cup \left(\bigcup_{q \in N_p^{r+1}} A_q|t \right)$$

and hence in some $A_x|t$ ($x = q$ or $x = p$) at the end of round r. Note that the edges (labeled $r+1$) added in round $r+1$ to A_p are irrelevant for $A_p|t$ here, since $t < r+1$.

Consequently, by the induction hypothesis, $(v \to w) \in G^t$, thereby confirming (i). As for (ii), the induction hypothesis also implies that $(v' \to w)$ is also in this $A_x|t$. Hence, every such edge must be in U and hence in $A_p|t$ at the end of round $r+1$ as asserted. □

The Lemma not only proves that $A_p|r$ is an underapproximation of G^r, but also that it is complete with respect to neighboring edges. It assert that if $A_p|r$ is s.c., then the component is a root component of G^r.

Lemma 7 ([11, Lem. 8]). *If the graph $C_p|s$ (line 15) with $s < r$ is non-empty in round r, then $p \in \mathcal{R}^s$.*

Proof. For a contradiction, assume that $C_p|s$ is non-empty (hence $A_p|s$ is an SCC), but $p \notin R^s$. Since p is always included in any A_p by construction and $A_p|s$ underapproximates G^s by Lemma 6.(i), this implies that $A_p|s$ cannot be the root component of G^s. Rather, $A_p|s$ must contain some process w that has an in-edge $(v \to w)$ in G^s that is not present in $A_p|s$. As w and hence some edge $(q \overset{s}{\to} w)$ is contained in $A_p|s$, because it is an SCC, Lemma 6.(ii) reveals that this is impossible. □

From this lemma and the description of inStableRoot(I) in Algorithm 1, we get the following Corollary 2. It rests on the fact that $A_p|s$ under approximates G^s (Lemma 6.(i)), but does so in a way that never omits an in-edge at any process $q \in A_p|s$ (Lemma 6.(ii)).

Corollary 2 ([11, Cor. 1])**.** *If the predicate inStableRoot(I) evaluates to true at process p in round r, then $\forall t \in I$ where $t < r$, it holds that $p \in R^t$.*

The following Lemma 8 in conjunction with Corollary 3 reveals that, in a sufficiently long interval of rounds $I = [r, s]$ with an I-vertex-stable root component R^I, every member p of R^I detects its membership in the $[r, s - D]$-VSRC R^I with a latency of at most D rounds (i.e., at the end of rounds $r + D$).

Lemma 8 ([11, Lem. 10])**.** *Consider an interval of rounds $I = [r, s]$, such that there is a D-bounded I-vertex-stable root component R^I and assume $|I| = s - r + 1 > D \geq \phi(R^I)$. Then, from round $r + D$ onwards, we have $C_p|r = R^I$, for every process in $p \in R^I$.*

Proof. Consider any $q \in R^I$. At the beginning of round $r + 1$, q has an edge $(q' \xrightarrow{T} q)$ in its approximation graph A_q with $r \in T$ iff $q' \in N_q^r$. Since processes always merge all graph information from other processes into their own graph approximation, it follows from the definition of a D-bounded I-vertex-stable root component in conjunction with the fact that $r+1 \leq s-D+1$ that every $p \in R^I$ has these in-edges of q in its graph approximation by round $r+1+D-1$. Since R^I is a vertex-stable root-component, it is strongly connected without in-edges from processes outside $\mathcal{R}^I$. Hence $C_p|r = R^I$ from round $r + D$ on, as asserted. □

Corollary 3 ([11, Cor. 2])**.** *Consider an interval of rounds $I = [r, s]$, with $|I| = s - r + 1 > D \geq D(R^I)$, such that there is a D-bounded vertex-stable root component R^I. Then, from the end of round s on, a call to inStableRoot$([r, s - D])$ returns R^I at every process in R^I.*

Together, Corollary 2 and Corollary 3 reveal that inStableRoot$(.)$ *precisely* characterizes the caller's actual membership in the $[r, s - D]$-VSRC R^I in the communication graphs from the end of round s on.

3.2 k-set agreement algorithm

We now explain how to solve k-set agreement based on the network approximation algorithm introduced in the previous section. The general idea is as follows: When a root component R is $2D + 1$-vertex-stable, in some interval $[r - 2D, r]$, all members *consistently* know in[1] round r that there existed a $D+1$-VSRC in $[r-2D, r-D]$ (cf. Corollary 3). They can hence *lock* on this event, by remembering a tuple (R, l, t, v) consisting of the *VSRC members* R, the *lockround* ℓ (which we choose as $\ell = r - 2D$), the round $t = r$ when the lock was created, and finally some *proposition value* v. Every process can obviously hold at most one such tuple, subsequently referred to as lock, in a round.

[1] That is, after the round r computing step of the network approximation algorithm, which is still before the end of round r of the k-set algorithm.

If a process holding a lock observes later (at latest D rounds after locking, i.e., $3D$ after the lockround $r - 2D$ stored in the current lock-tuple) that R was indeed a $2D + 1$-VSRC, it can decide on the value v in the lock-tuple and flood the network with the decision. This is feasible, since all members of R have the same lock tuple in this case: D-boundedness (recall Definition 3) guarantees that they have received *all* their round $r - 2D$ information from each other, in particular, their initial values, already by round $r - D$.

Unfortunately, however, locally, a process cannot know in round r whether the detected $D{+}1$-VSRC will indeed be stable for another D rounds. Our algorithm thus locks optimistically, possibly producing “unsuccessful” locks: In our model, it is not even possible to reliably find out whether some lock led to a decision at some process or not.

Our algorithm solves this problem by exchanging the full history of locks, both generated and learned later on, at all processes (described in detail below). In conjunction with the k-majority influence of Assumption 2, this ensures that if a $2D{+}1$-VSRC occurs, all members have *identical* lock histories (if we restrict our attention to locks created/learned by the start round $r - 2D$) and can hence agree on (the value of) one lock consistently, in a way that circumvents Theorem 5 in Chapter 4: (a) For the very first lock, we set the lock-tuple to $(R, r - 2D, r, v_{max})$, where v_{max} is just the maximum among all received initial values. (b) If there are at most k old locks in the system, we set the lock-tuple to one of those, selected consistently by means of some deterministic function. (c) If there are $> k$ old locks in the system, Assumption 2 guarantees that the same lock originating in a previous $2D + 1$-VSRC R_{cur} is chosen at all members in $R = R_{suc}$.

Algorithm 2 shows the code of the core k-set algorithm at process p_i, which uses the following variables: x_i denotes p_i’s proposal value, $r \geq 1$ is the current round, y_i returns p_i’s decision (initially $\perp$), and ℓ holds the current lockround ($\perp$ when no lock is currently set); D is the network causal diameter, which is in $\{1, \dots, n-1\}$.

A key component of the algorithm is the *lock history* structure $\texttt{hist}_i$, which is an array of sets of locks $\texttt{hist}_i[j]$ per process p_j in the system, with the following meaning: In round r,

- $\texttt{hist}_i[j]$ is the set of locks that p_i is sure that they are known by p_j so far,
- $\texttt{hist}_i[i]$ is the set of all locks created and learned by p_i so far.

In every round r, $\texttt{hist}_i$ is broadcast and updated according to $\texttt{hist}_j$ received from neighbors $p_j \in N^r_{p_i}$, using the following simple update rules:

(R) Remote process entry: $\forall p_m \in \Pi \backslash \{p_i\} : \texttt{hist}_i[m] := \texttt{hist}_i[m] \cup \big(\bigcup_{p_j \in N^r_{p_i}} \texttt{hist}_j[m]\big)$

(L) Local process entry: $\texttt{hist}_i[i] := \texttt{hist}_i[i] \cup \texttt{redate}\big(\bigcup_{j \in \Pi, j \neq i} \texttt{hist}_i[j] \backslash \texttt{hist}_i[i], r\big)$, where $\texttt{redate}(S, r)$ applied to a set S of locks and round r returns the set of locks obtained by replacing the lock creation time by r, formally, $\texttt{redate}(S, r) = \{(R, r_R, r, v) \mid (R, r_R, t, v) \in S\}$. Redating the locks, which can apply only to locks created at processes $\neq p_i$, allows p_i to remember the time when it learned first about a particular lock.

(N) New lock L created (of course locally): $\texttt{hist}_i[i] := \texttt{hist}_i[i] \cup \texttt{redate}(\{L\}, r)$. Note that a new lock may have been "derived" from a old lock, so $\texttt{redate}(\{L\}, r)$ stresses the fact that its learning time is always updated to r.

Evidently, the above update rules resemble the ones used for vector clocks [32]: The only process that can modify the creation/learning round in existing locks and create new locks (in $\texttt{hist}_i[i]$) is process p_i; any other process p_j can only add existing locks (that it hears about to exist in $\texttt{hist}_i[i]$) into $\texttt{hist}_j[i]$.

Keeping in mind this update history procedure, Algorithm 2 works as follows: After sending and receiving all round r messages, p_i analyzes the gathered information. If a decide message has been received, the algorithm decides on the received value (line 7–8). Otherwise, p_i updates its lock history $\texttt{hist}_i$ as described above, and then checks whether it is part of a $D+1$-VSRC. If this is not the case, the algorithm just goes to the next round. Otherwise, and if $\ell = \perp$, p_i locks by setting $\ell := r - 2D$ in line 17 and determining the lock tuple according to case (a), (b) and (c) above, in line 30, 28 and 25, respectively; the lock tuple is appended to its local lock data structure $\texttt{hist}[i]$, which represents (N) above.

If, on the other hand, $\ell \neq \perp$ and a $2D+1$-VSRC is detected, p_i can safely decide on the proposition value in the current lock tuple and terminate in line 34; in the terminal state, p_i does not further participate in the locking process but simply broadcasts its decision value. Should the VSRC eventually turn out not to live long enough to become a $2D+1$-VSRC, the lock is released in line 36.

Correctness proof

In this subsection, we will prove the following Theorem 1:

Theorem 1. *There exists an algorithm that solves k-uniform k-set agreement in a dynamic network that adheres to the conjunction of Assumption 1 and Assumption 2.*

The proof consists of a sequence of technical lemmas, which will finally allow us to establish all the properties of k-set agreement given in Chapter 2. Note that the claimed k-uniformity is obvious, as the code of the algorithm does not involve k. We start with simple properties related to generating locks at all members of vertex stable root components.

Lemma 9. *Only processes part of a vertex stable root with interval length greater than D (resp. $2D$) lock (resp. decide).*

Proof. The if in line 12 (resp. line 33) is only $true$ if inStableRoot detects a stable R in some interval of length $D+1$ (resp. of length $2D+1$) or larger, which implies by Corollary 2 that R is indeed a $D+1$-VSRC (resp. $2D+1$-VSRC). □

Lemma 10. *All processes part of a vertex stable root $R^{[r,s]}$ with interval length greater than $2D$, which did not start already before r, lock on $(R^{[r,s]}, r, r+2D, *)$ in round $r+2D$.*

Proof. Because $R^{[r,s]}$ is D-bounded by Assumption 1, Corollary 3 guarantees that inStableRoot$(r, r+D)$ returns R from round $r+2D$ (of the k-set-algorithm) on, and that it cannot have done so already in round $r+2D-1$. Hence, $\ell = \perp$ in round $r+2D$, so the algorithm must enter line 17 and create a lock in line 30, 28 or 25. In all cases, the lock is set to $(R^{[r,s]}, r, r+2D, *)$ as asserted. □

Lemma 11. *All processes part of a vertex stable root $R^{[r,s]}$ with interval length greater than $3D$, which did not start already before r, decide by round $r + 3D$.*

Proof. It follows from Lemma 10 that all members of the VSRC $R^{[r,s]}$ lock on $(R^{[r,s]}, r, r + 2D, *)$. As the VSRC remains stable also in rounds $r + 2D, \dots, r + 3D$, line 36 will not be executed in these rounds, thus $\ell = r$ remains unchanged. Consequently, due to Corollary 3, the if-statement in line 33 will evaluate to true at the latest in round $\ell + 3D = r + 3D$, causing all the processes to decide via line 34 by round $r + 3D$ as asserted. □

Although we know already that all members of a VSRC that is vertex stable for at least $3D$ rounds will decide, we did not prove anything about their decision values yet. In the sequel, we will prove that they decide on the *same* value.

Computing values rests on two sub-histories S_i, S'_i of $\mathtt{hist}_i$, which are computed when a lock is created for a $D + 1$-VSRC $R^{[r,s]}$ in round $r + 2D$, at process $p_i \in R^{[r,s]}$:

- The *bounded history* S_i, which throws away all locks created/learned after round r (and all virtual locks). It effectively reconstructs $\mathtt{hist}_i$ in lockround $\ell = r$.
- The *majority history* S'_i, which also throws away all locks in S_i from VSRCs that cannot exercise a majority influence on $R^{[r,s]}$. It effectively reconstructs $\mathtt{hist}_i$ at the end of the previous VSRC that exercises a majority influence on $R^{[r,s]}$ (if any).

The following Lemma 12 shows that, for all members of a VSRC $R^{r,s}$ of length at least $D + 1$, all entries in the bounded histories $S_i[j]$ (computed in round $r + 2D$) are identical to the lock histories $\mathtt{hist}^r_j[j]$ in round r. In what follows, we will ignore virtual locks, so equality of sets of locks like $S_i[j] = \mathtt{hist}_i[j]$ has to be interpreted modulo virtual locks, i.e., as $S_i[j] = \mathtt{hist}_i[j] \setminus \{(\emptyset, r, t, v) \mid (\emptyset, r, t, v) \in \mathtt{hist}_i[j]\}$.

Lemma 12. *Let $R^{[r,s]}$ be a VSRC with $s - r \geq D$ that does not start before r. Then, $\forall p, q \in R^{[r,s]}$: $S_p[q] = \mathtt{hist}^r_q[q]$. Therefore, $S_p = S_q$ and also $S'_p = S'_q$.*

Proof. In round r, $\mathtt{hist}^r_q[q] \supseteq \mathtt{hist}^r_p[q]$ according to the update rule (R). By Assumption 1 $R^{[r,s]}$ is D-bounded, thus there is a causal chain $(q \overset{r[k]}{\rightsquigarrow} p)$ of length at most D, along which p receives at least $\mathtt{hist}^r_q[q]$ and thus provides $\mathtt{hist}^{r+2D}_p[q] \supseteq \mathtt{hist}^r_q[q]$. No q in R can receive any messages from a process not in R, and every additional lock (R, r_R, t, v) created or learned (in the history update) at q in some round $r' > r$ (and hence put into $\mathtt{hist}^{r'}_q[q]$, from where it may reach $\mathtt{hist}^{r+2D}_p[q]$) has a creation/learning round $t = r'$. Since all these additional locks are thrown away when building $S_p[q]$, $S_p[q] = \mathtt{hist}^r_q[q]$ and hence also $S_p = S_q$ and $S'_p = S'_q$ follows as asserted. □

Lemma 13. *All decision values based on the same vertex stable root are equal.*

Proof. We have already established in Lemma 10 that each member of a vertex stable root $R^{[r,s]}$ that leads to a decision according to Lemma 11 locks on the same $(R^{[r,s]}, r, r + 2D, *)$. Since $S_p = S_q$ and $S'_p = S'_q$ by Lemma 12, all members of $R^{[r,s]}$ must set their lock using the *same* line 30, 28 or 25, which also implies identical lock and hence decision values v. □

We are now ready for proving the properties of k-set agreement.

Validity

Validity is straightforward to see as, at the start, the algorithm only considers his initial value for locking or deciding after a detected stable root (line 2 and 30). Moreover, any change of x_i in some lock involves an update via received locks from neighbors. But each neighbor p_j only sends locks containing its own value x_i or some updated value. Thus any lock value must be some initial value. Since the decision value y_i is based on some value in a lock (lines 34), validity holds.

Termination

Lemma 14. *The algorithm eventually terminates at all processes.*

Proof. For a contradiction, assume that there is $p \in \Pi$ which has not terminated after the stable interval guaranteed by Assumption 1. This implies that p is not part of a root component during this stable interval, because Lemma 11 ensures termination by $r_{GST} + 3D$ at latest for the latter. Hence, p did not get a decide message either. From Definition 4 of the E-bounded network, it follows that p must be causally influenced by some VSRC member within E rounds after its termination. Therefore, it must receive the decide message by $r_{GST} + 3D + E$ at latest. □

Agreement

For proving *k-Agreement*, we will combine the above lemmas with Assumption 2. In what follows, the last entry $l = (R, l, t, v)$ in a set of locks is a lock where t is maximal; in case of multiple locks with maximal t, the lexically smallest one, is returned.

Lemma 15. *Consider some VSRC $R^{[r,s]}$ with $s - r \geq 2D$ that does not start before r. In round $r + 2D$, the last lock l_p in $\mathtt{hist}_p^{r+2D}[p]$ of every member $p \in R^{[r,s]}$ is the same.*

Proof. Combining Lemmas 10 and 13, it follows that every member p creates the same lock l and adds it to $\mathtt{hist}_p[p]$ in round $\ell + 2D$. We claim that l is the last lock l_p in $\mathtt{hist}_p^{r+2D}[p]$. To see why, consider that Lemma 10 also implies that new locks created at p, i.e., locks which are not already in $\mathtt{hist}_p^{r}[p]$ in round r, can only be generated before round $r + D - 1$ (or after $s + 2D$): Such locks are based on a preceding $D + 1$-VSRC, which must end at round $r - 1$ at latest in order not to overlap with $R^{[r,s]}$; additional locks can only be generated after $R^{[r,s]}$ ends.

Therefore, $\mathtt{hist}_p^{r+D}[p] = \mathtt{hist}_p^{r+2D}[p]$, and by the same argument as in Lemma 12, it follows from the history update rule (R) that in fact even $\mathtt{hist}_q^{r+2D}[p] = \mathtt{hist}_p^{r+2D}[p]$ for all $q \in R^{[r,s]}$. Since no message from a process outside $R^{[r,s]}$ can arrive at any p, we have confirmed that l is indeed the latest lock in $\mathtt{hist}_p^{r+2D}[p]$. □

Lemma 16. *$R_{cur} \hookrightarrow_m R_{suc}$ guarantees that on the influenced processes $p \in |M| = IS(R_{cur}, R_{suc}) \subseteq R_{suc}$, $\exists q \in R_{cur} \forall q' \in R_{cur} : \mathit{hist}_p^{r_{suc}}[q'] = \mathit{hist}_q^{s_{cur}}[q'] \cup L$, where L are those new locks (created at q') process p learned about between round s_{cur} and r_{suc}. For the locks in L originating in VSRCs R that did not influence R_{cur} already, it holds that $|M'| = IS(R, R_{suc}) < |M|$.*

Proof. For every $p \in R_{suc}$ influenced by some process $q \in R_{cur}$, it holds by Definition 7 that $\forall q' \in R_{cur} : \texttt{hist}_p^{r_{suc}}[q'] = \texttt{hist}_q^{s_{cur}}[q'] \cup L$. This implies that the history of R_{cur} is embedded in the history of R_{suc}.

If we take a closer look, because of Definition 7, every additional information in L is originating in a VSRC $R \in \mathbb{V}_{D+1}$ that either (a) did not influence R_{cur} in which case we can drop it from L as it is already in $\texttt{hist}_q^{s_{cur}}[q']$, or else (b) did not influence R_{cur} in which case it can not overwhelm M as the resulting $|M'| =$ IS$(R, R_{suc}) < |M|$. □

Lemma 17. *$R_{cur} \hookrightarrow_m R_{suc}$ guarantees that $\forall p \in R_{suc}$, for a majority of $p' \in R_{suc}, \exists q \in R_{cur}\colon S'_p[p'] = hist_q^{s_{cur}}[q]$.*

Proof. Lemma 16 shows that there exists a set of processes $M \subseteq R_{suc}$ such that $\forall p' \in M \exists q \in R_{cur} \forall q' \in R_{cur}\colon \texttt{hist}_{p'}^{r_{suc}}[q'] = \texttt{hist}_q^{s_{cur}}[q'] \cup L$, which can not be overwhelmed by other locks. The update rule (L) implies that each $p \in M$ has to update $\texttt{hist}'_p[p']$ in such a way that $\texttt{hist}'_p[p']$ contains the information received from $\texttt{hist}_q^{s_{cur}}$, in particular, $\texttt{hist}_q^{s_{cur}}[q]$. Applying Lemma 12 yields that $\forall p \in R_{suc}\colon S_p[p'] = \texttt{hist}_q^{s_{cur}}[q] \cup L$. Finally, line 22 computes $S'_p[p']$ by removing all minority locks from S_p. Since all these minority locks with respect to $\texttt{hist}_q^{s_{cur}}[q]$ are in L, it follows that $S'_p[p'] = \texttt{hist}_q^{s_{cur}}[q]$ as asserted. □

Lemma 18. *$R_{cur} \hookrightarrow_m R_{suc}$ guarantees $\forall p, q, q' \in R_{suc}$ that the last lock in $S'_p[q]$ and $S'_p[q']$ is the same.*

Proof. Since Lemma 15 proves that all last entries at the end of R_{cur} are the same, and Lemma 17 shows that $\forall p, q \in R_{suc} \exists q' \in R_{cur}\colon \texttt{hist}_p[q] = \texttt{hist}_{q'}[q']$, we are done. □

Lemma 19. ***Majority Influence*** *guarantees that the lock in R_{suc} contains the same proposition value as the lock in R_{cur}.*

Proof. We know from Lemma 18 that the last entry in all entries in S'_p is the same $\forall p \in R_{suc}$. Hence the algorithm has to lock via case (C) (all supporter sets are the same and all entries are the same!) on every $p \in R_{suc}$. Thus, by Lemma 17, every $p \in R_{suc}$ locks on the the proposition value of the lock set in in R_{cur}. □

Lemma 20. *Assumption 2 guarantees a maximum of k different decision values during each run.*

Proof. From Lemma 19 we know that for $A, B \in \mathbb{V}_{2D+1}$, where $A \hookrightarrow_m B$ holds, B has to lock and subsequently decide on the lock of A. Hence if we combine Lemma 19 with Assumption 2 it follows that a maximum of k different decision exist on roots from $\mathbb{V}_{2D+1}$. By Lemma 9, only roots part of $\mathbb{V}_{2D+1}$ can decide, which concludes our proof. □

Algorithm 2 *k-Set Agreement (Code for Process p_i)*

```
Variables and Initialization:
$\texttt{hist}[*] := \emptyset$ // $\texttt{hist}[j]$ holds set of locks known by $p_j$
$\texttt{hist}[i] := \{(\emptyset, 1, 1, x_i)\}$ // virtual first lock ($R = \emptyset$) at $p_i$
$\texttt{decide} := \bot$, $\ell := \bot$ // most recent lock round, $\bot$ if none
Emit round r messages:
send $\langle \texttt{decide}, \texttt{hist} \rangle$ to all neighbors
Receive round r messages:
for all $p_j$ in $p_i$'s neighborhood $N^r_{p_i}$, receive $\langle \texttt{decide}_j, \texttt{hist}_j \rangle$ from $p_j$
Round r computation:
if $\texttt{decide} = \bot$ then
  if received $\texttt{decide}_j \neq \bot$ from some $p_j$ then
    $y_i := \texttt{decide}_j$, $\texttt{decide} := y_i$
  else
    Update lock history $\texttt{hist}$ from $\texttt{hist}_j, \forall p_j \in N^r_{p_i}$
    // Now get current root component, if any
    $C_i :=$ inStableRoot$(r - 2D, r - D)$
    if $C_i \neq \emptyset$ then
      $rs := r - 2D$ // $p_i$ is in $D + 1$-VSRC $C_i$, determine actual start round $rs$
      while inStableRoot$(r - 2D, r - D) =$ inStableRoot$(rs - 1, r - D)$ do
        $rs := rs - 1$
      if $\ell = \bot$ then
        $\ell := rs$ // Set new lock at $p_i$
        $S[*] := \emptyset$ // Determine bounded history $S$ (candidate locks)
        $\forall p_j \in C_i : S[j] := \{(R, r_R, t, v) \mid (R, r_R, t, v) \in \texttt{hist}[j],$ where $t \leq \ell$ and $R \neq \emptyset\}$
        $S'[*] := \emptyset$ // Determine majority history $S'$ of most frequent locks
        // macro $\#_{\texttt{Supt}}(s) = |\texttt{Supt}(s)|$ with $\texttt{Supt}(s) = \{m \mid \texttt{redate}(\{s\}, *) \in \texttt{redate}(S[m], *)\}$
        maj $:= \max\{\#_{\texttt{Supt}}(R, r_R, t, v) \mid (R, r_R, t, v) \in \bigcup_{j \in C_i} S[j]\}$
        $\forall p_j \in C_i : S'[j] := \{L \mid (L \in S[j]) \wedge (\#_{\texttt{Supt}}(L) = \text{maj})\}$
        if $\bigcup_{j \in C_i} S'[j] \neq \emptyset$ then
          // Check whether all latest locks in $S'$ are equal
          // macro $\texttt{last}(X)$ returns lock $(R, r_R, t, v) \in X$ with largest $t$ (lexically smallest if several)
          if for all $s, s' \in \bigcup_{j \in C_i} S'[j]$: $\texttt{Supt}(s) = \texttt{Supt}(s') = M$ and $\forall p, q \in M$: $\texttt{last}(S'[p]) = \texttt{last}(S'[q]) = \hat{s}$ then
            // Case (c): Lock to value of most recently learned common lock
            $\texttt{lock} := (C_i, \ell, r, \hat{s}.v)$
          else
            // Case (b): Take any lock in $S$, but choose it consistently
            // macro $\texttt{maxID}(C)$ returns largest process identifier of VSRC $C$
            $\hat{s} := (R, r_R, t, v) \in \bigcup_{j \in C_i} S[j]$ with $(\texttt{maxID}(R), r_R)$ lexical max
            $\texttt{lock} := (C_i, \ell, r, \hat{s}.v)$
        else
          // Case (a): Take value of any (virtual) lock in $\texttt{hist}$ consistently
          $\texttt{lock} := \Big(C_i, \ell, r, \max\big\{v \mid (R, r_R, t, v) \in \bigcup_{j \in C_i} \texttt{hist}[j]\big\}\Big)$
        // Locking done, so put newly created lock into lock history
        $\texttt{hist}_i[i] := \texttt{hist}_i[i] \cup \{\texttt{lock}\}$
      else
        if inStableRoot$(\ell, \ell + 2D) \neq \emptyset$ then
          $y_i := \texttt{lock}.v$, $\texttt{decide} := y_i$
    else
      $\ell := \bot$
```

CHAPTER 4

Impossiblity results

In this section[1] , we provide two impossibility results for solving k-set agreement in our model, under two strong assumptions constraining the adversary, which are natural generalizations of the consensus case [10]. They disproved our initial conjecture that these constraints might be sufficient for solving general k-set agreement. We first show the necessity of at least some concurrent stability of the k root components (cf. Section 4.2). Essentially, this is achieved by deriving a violation of the termination condition. Second, we prove that even if, from some round on, the k root components become static forever, k-set agreement cannot be solved (cf. Section 4.3). In contrast to the first impossibility, the second one is obtained by constructing an execution where the k-agreement property is violated.

4.1 Definitions from [7]

We first introduce the detailed definitions needed for the proof technique introduced in [7]:

Definition 9 (Indistinguishable Executions)**.** *Two executions of an algorithm α, β are* indistinguishable *for a set of processes D, denoted $\alpha \overset{D}{\sim} \beta$, if for any $p \in D$ it holds that p executes the same state transitions in α and in β.*

Now consider a model of a distributed system $\mathcal{M} = \langle \Pi \rangle$ that consists of the set of processes Π and a *restricted model* $\mathcal{M}' = \langle D \rangle$ that is computationally compatible to $\mathcal{M}$ (i.e., an algorithm designed for a process in $\mathcal{M}$ can be executed on a process in $\mathcal{M}'$) and consists of the set of processes $D \subseteq \Pi$. The restriction of an algorithm for this setting is defined as follows:

Definition 10 (Restriction of an Algorithm from [7, Definition 1])**.** *Let $\mathcal{A}$ be an algorithm that works in system $\mathcal{M} = \langle \Pi \rangle$ and let $D \subseteq \Pi$ be a nonempty set of processes. Given any restricted*

[1] The material provided in this section was devised jointly with Kyrill Winkler, hence can also be found in a slightly different form in his Master thesis [42]

system $\mathcal{M}' = \langle D \rangle$, the restricted algorithm $A_{|D}$ *for system $\mathcal{M}'$ is constructed by dropping all messages sent to processes outside D in the message sending function of $\mathcal{A}$.*

We also need the following similarity relation between runs in computationally compatible systems (cf. [7, Definition 3]):

Definition 11 (Compatibility of runs from [7, Definition 3]). *Let $\mathcal{R}$ and $\mathcal{R}'$ be sets of runs, and D be a non-empty set of processes. We say that* runs $\mathcal{R}'$ are compatible with runs $\mathcal{R}$ for processes in **D**, *denoted by $\mathcal{R}' \preccurlyeq_D \mathcal{R}$, if $\forall \alpha \in \mathcal{R}' \; \exists \beta \in \mathcal{R}\colon \alpha \overset{D}{\sim} \beta$.*

Theorem 2 (k-Set Agreement Impossibility [7, Thm. 1]). *by some fixed algorithm A in $\mathcal{M}$, where every process starts Let $\mathcal{M} = \langle \Pi \rangle$ be a system model and consider the runs $\mathcal{M}_A$ that are generated with a distinct input value. Fix some nonempty and pairwise disjoint sets of processes $D_1, \ldots, D_{k-1}$, and a set of distinct decision values $\{v_1, \ldots, v_{k-1}\}$. Moreover, let $D = \bigcup_{1 \le i < k} D_i$ and $\overline{D} = \Pi \setminus D$. Consider the following two properties:*

(dec-D) *For every set D_i, value v_i was proposed by some $p \in D$, and there is some $q \in D_i$ that decides v_i.*

(dec-$\overline{D}$) *If $p_j \in \overline{D}$ then p_j receives no messages from any process in D until every process in $\overline{D}$ has decided.*

Let $\mathcal{R}_{(\overline{D})} \subseteq \mathcal{M}_A$ and $\mathcal{R}_{(D,\overline{D})} \subseteq \mathcal{M}_A$ be the sets of runs of A where (dec-$\overline{D}$) respectively both, (dec-D) and (dec-$\overline{D}$), hold.[2] *Suppose that the following conditions are satisfied:*

(A) *$\mathcal{R}_{(\overline{D})}$ is nonempty.*

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$.

In addition, consider a restricted model $\mathcal{M}' = \langle \overline{D} \rangle$ such that the following hold:

(C) *There is no algorithm that solves consensus in $\mathcal{M}'$.*

(D) $\mathcal{M}'_{A_{|\overline{D}}} \preccurlyeq_{\overline{D}} \mathcal{M}_A$.

Then, A does not solve k-set agreement in $\mathcal{M}$.

Moreover, we will rely on the following consensus impossibility result from [10]:

Assumption 3 (Consensus assumption [10, Assumption 4]). *For any round r, there is exactly one root component R^r in G^r. Moreover, there exists a D and an interval of rounds $I = [r_{ST}, r_{ST} + D - 2]$, such that there is an I-vertex stable root component R^I, and there exists a unique $q \in \Pi$ such that $\forall p \in R^I, \forall r \in I : cd^r(p,q) \le D$, while for all $q' \in \Pi \setminus \{q\}$ we have $\forall p \in R^I, \forall r \in I : cd^r(p,q') \le D - 1$.*

Theorem 3 (Consensus impossibility [10, Thm. 5]). *Assume that Assumption 3 is the only requirement for the graph topologies. Then consensus is impossible.*

[2] Note that $\mathcal{R}_{(\overline{D})}$ is by definition compatible with the runs of the restricted algorithm $A_{|\overline{D}}$.

4.2 Impossibility assuming $\leq k$ simultaneous roots

The first natural attempt to make k-set agreement solvable in our model was to restrict the maximum number of root components per round to k:

Assumption 4. $\forall r > 0$, *G^r contains at most k root components.*

Unfortunately, Assumption 4 is too weak for solving k-set agreement: Theorem 4 shows that even k root components which are vertex-stable simultaneously for up to $n - k - 1$ rounds are not sufficient.

Theorem 4. *There exists no algorithm that solves k-set agreement with $n > k + 1$ nodes under Assumption 4, for any $1 \leq k < n$, even if there are $k - 1$ root components R_1 to R_{k-1} that are vertex-stable forever and one root component R_k is vertex-stable for at most $n - k - 1$ rounds.*

For $k = 1$, Theorem 4 is eqivalent to [10, Thm. 4]. The proof for $k > 1$ is based on the existence of a forever bivalent consensus run in a suitably chosen restricted system, which violates the termination condition of k-set agreement.

Proof. Let $D_i = p_i$ for $0 < i \leq k - 1$. Consequently, $\overline{D} = \{p_k, p_{k+1}, \ldots, p_n\}$ and $|\overline{D}| \geq 2$. Suppose that there is a k-set algorithm $\mathcal{A}$. We will prove that the conditions of Theorem 2 (Theorem 1 of [7]) are satisfied; thus providing a contradiction to the assumption that $\mathcal{A}$ exists.

(A) $\mathcal{R}_{(\overline{D})}$ is nonempty: In order to allow an execution where the processes in $\overline{D}$ do not receive any messages from processes not in $\overline{D}$, the communication graph cannot have any incoming links to $\overline{D}$ from D until every process in $\overline{D}$ has decided. Clearly such a communication graph exists and satisfies the assumptions of the theorem.

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{\mathbf{D}}} \mathcal{R}_{(D,\overline{D})}$: Let $\mathcal{H}$ be the set of runs where processes have unique input values $x_i = i$, $0 < i \leq n$. Moreover, assume that, in all runs in $\mathcal{H}$, the communication graphs are such that $p_1, \ldots, p_{k-1}$ are disconnected and $p_k, \ldots, p_n$ are weakly connected until every process has decided.

By definition, any run in $\mathcal{R}_{(\overline{D})}$ has a communication graph in every round with no links from the set of processes $D = \bigcup_{i=1}^{k-1} D_i$ to the set of processes $\overline{D}$. The additional restriction imposed on the communication graph of the executions $\in \mathcal{H}$ is that there are no links from the set of processes $\overline{D}$ to the set of processes D until decision. Note that the presence or absence of these links can have no influence on the state transitions of the processes in $\overline{D}$, because the only way processes in $\overline{D}$ could learn about the status of such a link would be via an incoming link from some process $\in D$. Thus, for every $\rho \in \mathcal{R}_{(\overline{D})}$, there exists a $\rho' \in \mathcal{H}$ such that the processes in $\overline{D}$ make the same state transitions in ρ and in ρ' until deciding. Moreover, $\mathcal{H} \subseteq \mathcal{R}_{(D,\overline{D})}$, thereby establishing $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$.

(C) Consensus is impossible in $\mathcal{M}' = \left\langle \overline{\mathbf{D}} \right\rangle$: Suppose that we have a perpetually changing single root component, except for at least one interval of rounds $I = [r_{ST}, r_{ST} + D - 2]$, where $D = n - k$, for some fixed r_{ST}. During this interval, let the topology of $\overline{D}$ be a chain with head p and tail q. This satisfies our initial restriction, with p being the vertex-stable root component that lasts for $n - k - 1$ rounds.

Since $|\overline{D}| = n - k + 1 \geq 3$, this chain has length $D = n - k \geq 2$, which means that for all rounds $r \in I$, it holds that $\mathtt{cd}^r(p, q) = D$ and $\forall q' \in \overline{D} \setminus \{q\} \colon \mathtt{cd}^r(p, q') < D$. Since our assumptions for $\overline{D}$ are the same as made in Assumption 3 with $D = n - k$, by Theorem 3, consensus is impossible in $\overline{D}$.

(D) $\mathcal{M}'_{\mathcal{A}_{|\overline{\mathbf{D}}|}} \preccurlyeq_{\overline{\mathbf{D}}} \mathcal{M}_{\mathcal{A}}$: Fix any run $\rho' \in \mathcal{M}'$ and consider the run $\rho \in \mathcal{M}$ where every process in $\overline{D}$ has the same sequence of states in ρ as in ρ'. By the properties of $\mathcal{M}$, the processes not in $\overline{D}$ can be disconnected in every round of ρ, yielding $\rho \overset{\overline{D}}{\sim} \rho'$. □

4.3 Eventual stability does not help

The result of Theorem 4 suggested that at most k root components per round that are also eventually vertex-stable forever might be sufficient for solving k-set agreement. Unfortunately, this is not the case: Theorem 5 shows that it is possible to construct an execution where the k-agreement property is violated.

Theorem 5. *Suppose that in every run there is a* stabilization round r_{GST} *such that, for all $r \geq r_{\text{GST}}$, it holds that $G^r = G^{r+1}$ and there are no other restrictions on the communication graphs apart from Assumption 4. Then, there is no algorithm that solves k-set agreement for $1 < k < n$.*

Proof. We start our proof with some notation and technical lemmas. For some model $\mathcal{M}$ and some algorithm $\mathcal{A}$, we denote by $\mathcal{M}_{\mathcal{A}}$ the set of runs of algorithm $\mathcal{A}$ on $\mathcal{M}$. Let $\mathcal{M} = \langle \Pi \rangle$ be our system with $|\Pi| > 2$ that is restricted by the assumptions of the theorem, and let $D = \{p_1, p_2\} \subseteq \Pi$. We consider the restricted model $\mathcal{M}' = \langle D \rangle$ and the restricted algorithm $A_{|D}$ of algorithm $\mathcal{A}$ on D. Except for the number of processes, $\mathcal{M}'$ has the same properties as $\mathcal{M}$, except that $\mathcal{M}'$ guarantees a single root component in every round.

Following the generic impossibility proof of [7], we will argue that if there was a correct k-set agreement algorithm $\mathcal{A}$ for $\mathcal{M}$, then the restriction $A_{|D}$ would solve consensus when being run on $\mathcal{M}'$: Since the assumption of k root components per round allows G^r to partition into k isolated root partitions, there are executions in $\mathcal{M}_{\mathcal{A}}$ where the processes in D receive no messages from any process outside of D, and decide on a unique value value in every root partition. On the other hand, when executing $A_{|D}$ on $\mathcal{M}'$, the processes of D clearly also receive no messages from any process of $\Pi \setminus D$. Thus, the processes of D cannot distinguish whether they execute $A_{|D}$ on $\mathcal{M}'$ or $\mathcal{A}$ on $\mathcal{M}$, and must hence also agree on a single value. Note that, because $n = |D| = 2$ in $\mathcal{M}'$, we can re-use classic bivalency arguments since there are at most two initial values (although we assume $|V| > k$).

Lemma 21. *For every k-set agreement algorithm $\mathcal{A}$ for $\mathcal{M}$, there exists a run $\rho' \in \mathcal{M}'_{A_{|D}}$ that is bivalent for all rounds up to and including r_{GST}.*

Proof. Similar to the strategy used in [38] to show the undecidability with lossy links, our proof proceeds by induction.

For the base case consider the initial configuration $C^1(x_1, x_2)$, where p_1 starts with initial value x_1 and p_2 starts with x_2 and $x_1 \neq x_2$. Assume that $C^1(x_1, x_2)$ is univalent. By Validity the only possible decision values in runs starting from this initial configuration are x_1 and x_2. In order to see that $C^1(x_1, x_2)$ is neither x_1-valent nor x_2-valent (and hence bivalent) we show that $C^1(x_1, x_2)$ is not x_1-valent (the case of x_2-valency follows from the symmetric argument). Consider a run starting from $C^1(x_1, x_2)$ where $\forall r > 0\colon (p_1 \to p_2) \notin G^r$. This run is indistinguishable to p_2 from a run with the same communication graphs but starting from $C^1(x', x_2)$, with $x' \neq x_1$. By Validity, p_2 cannot decide x_1 in the latter run, showing that $C^1(x_1, x_2)$ cannot be x_1-valent.

For the inductive step, we show that if C^{r-1} is bivalent then there is a bivalent successor configuration C^r of C^{r-1} that is bivalent or $r - 1 > r_{\text{GST}}$. If $r - 1 > r_{\text{GST}}$ we are done, so we assume that $r - 1 \leq r_{\text{GST}}$ for the remainder of this proof. This allows us to freely choose G^{r-1}.

Assume that all C^r are univalent. As the successor configurations of C^{r-1} are uniquely determined by the round graph G^{r-1} and because of the assumption that there is a single root component, we need to consider only three successor configurations. Let C^r_{01} be the successor configuration of C^{r-1} that is reached by the G^{r-1} with $E^{r-1} = \{(p_1 \to p_2)\}$, let C^r_{10} be the successor configuration of C^{r-1} that is reached by the G^{r-1} with $E^{r-1} = \{(p_1 \leftarrow p_2)\}$, and let C^r_{11} be the successor configuration of C^{r-1} that is reached by the G^{r-1} with $E^{r-1} = \{(p_1 \to p_2), (p_1 \leftarrow p_2)\}$. As all C^r are univalent, w.l.o.g. assume that C^r_{11} is x_1-valent. Because C^{r-1} is bivalent, at least one of C^r_{10}, C^r_{01} must be x_2-valent; w.l.o.g. assume that C^r_{10} is x_2-valent. Note that the only difference between C^r_{11} and C^r_{10} is that p_2 received p_1's message in the former but not in the latter. Consider now the executions starting from C^r_{10}, respectively C^r_{11}, where it holds that $\forall r' > r\colon (p_1 \leftarrow p_2) \notin G^{r'}$. Both executions are indistinguishable for p_1 because p_2 can never tell p_1 whether p_2 received p_1's round r' message. Since p_1 must eventually decide the same in both executions, they cannot have different valences. □

Lemma 22. *For every algorithm $\mathcal{A}$ for $\mathcal{M}$, the set of runs $\mathcal{R} \subseteq \mathcal{M}_A$ where G^r contains arbitrary outgoing edges from D but no incoming edges to D satisfies $\mathcal{M}'_{A|_D} \preccurlyeq_D \mathcal{R}$.*

Proof. By the assumptions of $\mathcal{R}$, the processes in D never receive messages from any process of $\Pi \setminus D$. Therefore, in any run of $\mathcal{R}$, the state transitions of the processes in D cannot depend on the state of any process of $\Pi \setminus D$. This establishes $\mathcal{M}'_{A|_D} \preccurlyeq_D \mathcal{R}$. □

We are now ready to prove Theorem 5. Our proof relies on an execution, where every k-set agreement algorithm with $n > 2$ and $1 < k < n - 1$ produces $k + 1$ decisions The run is constructed as follows (cf. Figure 4.2): For each p_i, we choose a unique proposal value x_i such that x_1 and x_2 are in accordance with Lemma 21.

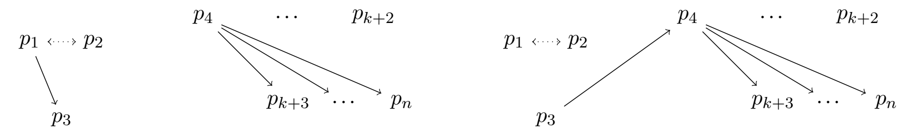
For the rounds $1 \leq r \leq x$, where x is chosen as described below, we use a G^r constructed as follows (cf. Figure 4.1a):

- p_1, p_2 are connected to each other as in the bivalent run ρ' provided by Lemma 21, and have no incoming edges from any $\Pi \setminus D$.

- p_3 has an incoming edge only from p_1 and no outgoing edges.

- $p_4, \ldots, p_{k+2}$ form single-node root components.

- The remaining processes (if any) have an incoming edge from p_4 but no outgoing edges.



(a) One (changing) root component among p_1 and p_2, single root p_4, $k-2$ remaining single-node root components $p_5, \ldots, p_{k+1}$ among $p_5, \ldots, p_n$.

(b) One root component among p_1 and p_2, single root p_3, $k-2$ remaining single-node root components $p_5, \ldots, p_{k+1}$ among $p_5, \ldots, p_n$.

Figure 4.1: Communication graphs used in the proof of Theorem 5: (a) depicts G^r for $0 < r \leq x$, while (b) depicts G^r for $x < r \leq y$. The dotted edge indicates an unstable ("moving") link between p_1 and p_2.

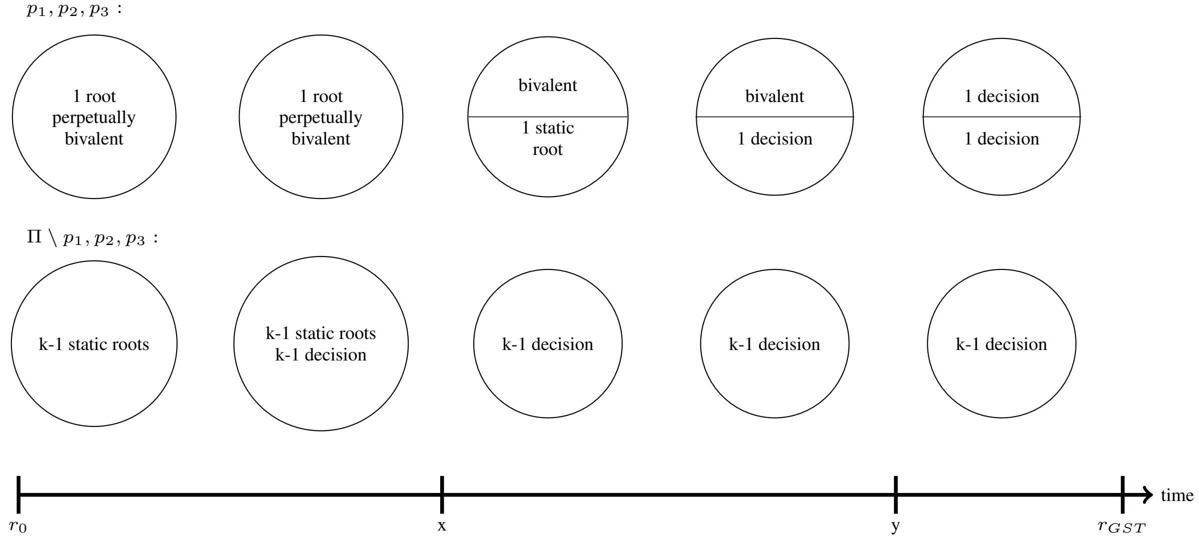


Figure 4.2: Example execution for the withholding argument

Note that this graph contains one root component among p_1, p_2, p_3 and $k-1$ root components among $p_4, \ldots, p_n$, thereby satisfying Assumption 4. A simple indistinguishability argument shows, that there is some finite round x, s.t. processes $p_4, \ldots, p_{k+2}$ have decided on $k-1$ distinct values: Just consider the execution where p_1 is a perpetual root among p_1, p_2, p_3 and $p_4, \ldots, p_n$ are as defined above perpetually. Since we assumed a correct algorithm, there must be an x such that all processes $p_4, \ldots, p_{k+2}$ decide on some value before x. To see that there are $k-1$ different decision observe that $p_4, \ldots, p_{k+2}$ never learn a value that is not their own initial value, thus this fact follows from validity.

From round $x+1$ on, we use a communication graph G^r that is the same as above, except that the edge from p_1 to p_3 is removed and an edge from p_3 to p_4 is added (cf. Figure 4.1b). Note carefully that the total number of root components is preserved, and that there are still no incoming edges to $\{p_1, p_2\}$. Since the resulting execution is admissible in $\mathcal{M}$, by a similar reasoning as above, there must be some round y s.t. p_3 decides before y; its decision value must be in $\{x_1, x_2, x_3\}$, since p_3 can have heard at most from $\{p_1, p_2, p_3\}$. Obviously, p_1 and p_2 are still undecided.

Finally, it follows from the bivalent configuration of $A_{|D}$ reached by round y, according to Lemmas 21 and 22, that there exist communication graphs Γ_1 resp. Γ_2 for all rounds $r > y$, which are the same as the graphs used for rounds $x < r \leq y$, except that the links between p_1 and p_2 are chosen such that they both decide on x_1 resp. x_2. We now continue our execution with Γ_2 if p_3 decided x_1, and Γ_1 otherwise. Obviously, this guarantees that p_1, p_2 and p_3 reach at least two different decisions.

As we have now reached a total of $k + 1$ decisions, we have established a contradiction. This completes the proof of Theorem 5. □

CHAPTER 5

Conclusions

5.1 Summary of accomplishments

In this Master thesis, we defined a model for solving k-set agreement in dynamic synchronous networks. Using the concept of vertex stable root components (VSRC), i.e., strongly connected components without incoming edges, with invariant members but possibly changing interconnections, it assumes two properties, namely, majority influence (causal influence between certain VSRCs) and an eventually stable interval (concurrently stable VSRCs). According to the classification in [17], the proposed model is rather weak in comparison to other models (stronger than class 1, but strictly weaker than class 3).

We also proposed a k-set agreement algorithm for this model, which uses a network approximation for detecting VSRCs and forcing their members to decide if they are stable sufficiently long. The algorithm is k-uniform, i.e., does not know k, hence the number of decision values is adjusted automatically to the actual network connectivity in a run. Such a feature is interesting for systems that can still operate with such a form of graceful degradation. A comprehensive correctness proof showed that k-set agreement is indeed possible in our model.

On the other hand, we showed that more than k simultaneous VSRCs, as well as at most k root components that are not sufficiently long vertex-stable, render k-set agreement impossible. Therefore the properties used in our model are close to the solvability border.

5.2 Discussion of our results

Model

Whereas the model introduced for consensus in [10] adequately captures the behavior of a dynamic network with unidirectional links in general, the connectivity assumptions were too restrictive to be useful for k-set agreement with $k > 1$. Therefore, our relaxation to allow several weakly connected components/roots was necessary to construct a useful basic model for the problem at hand. Note that the model in this thesis has the same difficulty in regard to finding a suitable causal diameter D and E as the model of [10].

Just using the model and algorithm of [10] with the relaxations of at most k VSRCs, it is possible for a single process to hop between subgraphs, and thus hiding locks and even decisions long enough to violate agreement. As Chapter 4 proves, if we want to use the locking principle to solve k-set agreement, we need two additional assumptions, namely, majority influence and stable interval.

Compared to [9], the resulting model is far more dynamic, thus connectivity between single processes is not as restricted. Especially, [9] forces a few edges (at least $n-1-k$) to be constantly active throughout the whole run of the system, which has poor coverage, for example, in wireless systems.

Majority influence: As Theorem 5 proves, the model needs some kind of information propagation with respect to already made decisions. Thus it seems natural to impose a model property that makes sure that locks possibly leading to decisions are carried over to future components. Moreover, because only unidirectional connections exists between non-root nodes and root nodes, it is necessary to guarantee some influence between such "important" locks and the root of a newly formed component.

Although the majority influence property given in Assumption 2 may appear somewhat contrived, it is natural w.r.t. the weak guarantees on information propagation that can be given in our model under Assumption 1. Moreover, one readily figures out more natural stronger assumptions that imply Assumption 2:

(i) Replacing majority influence $\mathrm{IS}(R_{cur}, R_{suc})$ in Definition 7 by majority intersection $|R_{suc} \cap R| < |R_{suc} \cap R_{cur}|$, which is obviously the strongest form of influence.

(ii) We could even require $|R_{suc} \cap R_{cur}| > |R_{suc}|/2$, i.e., a majority intersection with respect to the number of processes in R_{suc}. This could be interpreted as a changing VSRC, in the sense of "R_{suc} is the result of changing a minority of processes in R_{cur}". Although this restricts the rate of growth of VSRCs in a run, it would apply, for example, in case of random graphs where the giant component has formed [20, 28].

Notice however, that we do not argue that the particular stability intervals used in our model are the optimum, but rather that the idea of some form of majority influence is needed for every algorithm to work.

In particular, we cannot argue that connecting $2D + 1$-VSRCs is necessary. It may be possible to use k-majority influence in conjunction with VSRCs with shorter intervals and still prevent a safety violation. However, preliminary studies of consensus in a similar model suggest that $2D + 1$ may indeed be a lower bound also for k-set agreement.

In any case , the stability intervals used in this thesis are "adjusted" to the algorithm at hand, but can be modified to fit any other solution with different decision intervals. Assume that we have an algorithm executed on a node, which can, based on some kind of information gathering, decide on some value. Because of the dynamic nature of the network, we can never know how many other nodes are influenced by this decision or some previous proposal of the deciding node. This shows the fundamental problem that arises when solving k-set agreement, in a dynamic networks. Thus, to ensure that agreement is not violated, we have to somehow capture the first k decisions and make sure that later decisions are based on the earlier ones, which is accomplished by the majority influence property in Assumption 2.

Stable interval: We notice that the stable influence property is an enhanced version of the termination criteria of [10], where a $4D + 1$ interval is needed to guarantee termination. For k-set agreement, it is a reasonable assumption that an interval of similar length with k stable roots is needed to ensure termination as well. Indeed we already provided a proof that shows that an interval with k concurrent vertex stable roots of some length X is needed to guarantee termination. However, we do not know whether X is tight: X has to be long enough such that all connected nodes can be reached by a deciding node, but it might even be longer.

One could argue that there exist other means to ensure termination of a process. However, independent of this local termination, it also has to ensure global termination. It seems difficult to imagine an approach different from the stable interval property, that can guarantee this.

Algorithm

The **network approximation algorithm** is nearly identical to the one used in [10]. The sole change is that the information delivered by the inStableRoot is not simply $True$ or $False$ but the members of the root R itself.

The `hist` structure used in the **agreement algorithm** resembles *vector clocks*: Each process tries to gather the latest information from every other process in the network. The main difference to vector clocks is that each node remembers not only the last entry from its neighbors, but saves the complete history from each participant. Hence, if some process p_i sends a message to p_j during the run, p_j's `hist` contains p_i's `hist` until the send round. This knowledge, in combination with the k-majority influence property, enables each process to establish a "happens-before" relation between locks, which is equivalent to the relation between events of a vector clock.

5.3 Possibilities for future work

An important open question is model coverage, i.e., the question whether the stability conditions needed to solve k-set agreement can be achieved in existing wireless networks. But this exceeds the scope of this thesis which takes a purely theoretical approach to the problem. Suitable simulation experiments and the resulting data may be the topic of another thesis.

Moreover, the border between the majority influence property and the related impossibility results is not tight. Thus, research is needed to improve either the possibility or the impossibility or maybe even both.

Lastly, the algorithm was not designed for memory efficiency, thus optimizing memory usage to increase its usability in embedded applications is still a goal that needs to be accomplished.

Bibliography

[1] Yehuda Afek, Eli Gafni, and Adi Rosen. The slide mechanism with applications in dynamic networks. In *ACM PODC*, pages 35–46, 1992.

[2] D. Alistarh, S. Gilbert, R. Guerraoui, and C. Travers. Of choices, failures and asynchrony: The many faces of set agreement. In *ISAAC 2009*, pages 943–953. Springer, Heidelberg, 2009.

[3] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 290–299, New York, NY, USA, 2004. ACM.

[5] Baruch Awerbuch and Shimon Even. Efficient and reliable broadcast is achievable in an eventually connected network(extended abstract). In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, pages 278–281, New York, NY, USA, 1984. ACM.

[6] Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Michael E. Saks. Adapting to asynchronous dynamic networks. In *STOC'92*, pages 557–570, 1992.

[7] Martin Biely, Peter Robinson, and Ulrich Schmid. Easy impossibility proofs for k-set agreement in message passing systems. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC '11, pages 227–228, New York, NY, USA, 2011. ACM.

[8] Martin Biely, Peter Robinson, and Ulrich Schmid. Easy impossibility proofs for k-set agreement in message passing systems. *CoRR*, abs/1103.3671, 2011.

[9] Martin Biely, Peter Robinson, and Ulrich Schmid. Solving k-set agreement with stable skeleton graphs. In *IPDPS Workshops*, pages 1488–1495, 2011.

[10] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In *19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS, 2012. (to appear).

[11] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. *CoRR*, abs/1204.0641, 2012.

[12] Martin Biely, Josef Widder, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, and André Schiper. Tolerating corrupted communication. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, PODC '07, pages 244–253, New York, NY, USA, 2007. ACM.

[13] Elizabeth Borowsky and Eli Gafni. Generalized flp impossibility result for t-resilient asynchronous computations. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 91–100, New York, NY, USA, 1993. ACM.

[14] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Deterministic Computations in Time-Varying Graphs: Broadcasting under Unstructured Mobility. In *Proc. of 5th IFIP Conference on Theoretical Computer Science (TCS)*, 2010.

[15] Arnaud Casteigts, Serge Chaumette, and Afonso Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proceedings of the 16th international conference on Structural Information and Communication Complexity*, SIROCCO'09, pages 126–140, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] Arnaud Casteigts, Paola Flocchini, Bernard Mans, and Nicola Santoro. A strict hierarchy of dynamic graphs for shortest, fastest, and foremost broadcast. *CoRR*, abs/1210.3277, 2012.

[17] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *CoRR*, abs/1012.0009, 2010.

[18] Bernadette Charron-Bost and André Schiper. The heard-of model: Computing in distributed systems with benign failures. In *Distributed Computing*, pages 22(1):49—-71, April 2009.

[19] Soma Chaudhuri. More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems. *Information and Control*, 105(1):132–158, July 1993.

[20] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin. Giant strongly connected component of directed networks. *Phys. Rev. E*, 64:025101, Jul 2001.

[21] A. Ferreira. Building a reference combinatorial model for manets. *Netwrk. Mag. of Global Internetwkg.*, 18(5):24–29, September 2004.

[22] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[23] 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*, pages 143–152. ACM Press, 1998.

[24] Alois Goiser, Samar Khattab, Gerhard Fassl, and Ulrich 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, June 2010.

[25] Fabíola Greve, Luciana Arantes, and Pierre Sens. What model and what conditions to implement unreliable failure detectors in dynamic networks? In *Proceedings of the 3rd International Workshop on Theoretical Aspects of Dynamic Distributed Systems*, TADDS '11, pages 13–17, New York, NY, USA, 2011. ACM.

[26] Ted Herman and Chen Zhang. Best paper: stabilizing clock synchronization for wireless sensor networks. In *Proceedings of the 8th international conference on Stabilization, safety, and security of distributed systems*, SSS'06, pages 335–349, Berlin, Heidelberg, 2006. Springer-Verlag.

[27] Stephan Holzer, Yvonne Anne Pignolet, Jasmin Smula, and Roger Wattenhofer. Monitoring churn in wireless networks. *Theor. Comput. Sci.*, 453:29–43, September 2012.

[28] Svante Janson, Donald E. Knuth, Tomasz Luczak, and Boris Pittel. The birth of the giant component. *Random Struct. Algorithms*, 4(3):233–359, 1993.

[29] Fabian Kuhn, Rotem Oshman, and Yoram Moses. 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, New York, NY, USA, 2011. ACM.

[30] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[31] Igor Litovsky, Yves Métivier, and Éric Sopena. Handbook of graph grammars and computing by graph transformation. chapter Graph relabelling systems and distributed algorithms, pages 1–56. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[32] Friedemann Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[33] D. L. Mills. Network time protocol (ntp), 1985.

[34] Regina O'Dell and Rogert Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the 2005 joint workshop on Foundations of mobile computing*, DIALM-POMC '05, pages 104–110, New York, NY, USA, 2005. ACM.

[35] Ram Ramanathan, Prithwish Basu, and Rajesh Krishnan. Towards a formalism for routing in challenged networks. In *Proceedings of the second ACM workshop on Challenged networks*, CHANTS '07, pages 3–10, New York, NY, USA, 2007. ACM.

[36] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: the topology of public knowledge. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 101–110, New York, NY, USA, 1993. ACM.

[37] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '89, pages 304–313, London, UK, UK, 1989. Springer-Verlag.

[38] Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM J. Comput.*, 38(5):1912–1951, January 2009.

[39] M. Schwarz, K. Winkler, M. Biely, P. Robinson, and U. Schmid. k-set agreement in dynamic networks. In *Proceedings OPODIS,subbmitted*, 2013.

[40] M. Schwarz, K. Winkler, M. Biely, P. Robinson, and U. Schmid. k-set agreement in dynamic networks. 2013.

[41] William Su, Sung-Ju Lee, and Mario Gerla. Mobility prediction and routing in ad hoc wireless networks. *Int. J. Netw. Manag.*, 11(1):3–30, January 2001.

[42] Kyrill Winkler. Easy impossibility proofs for k-set agreement. In *Master Thesis*, 2013.