Towards a Universal Approach for Monotonic Searchability in Self-Stabilizing Overlay Networks [*]

Christian Scheideler, Alexander Setzer, and Thim Strothmann

Paderborn University, Germany
{scheidel,asetzer,thim}@mail.uni-paderborn.de

Abstract. For overlay networks, the ability to recover from a variety of problems like membership changes or faults is a key element to preserve their functionality. In recent years, various self-stabilizing overlay networks have been proposed that have the advantage of being able to recover from *any* illegal state. However, the vast majority of these networks cannot give any guarantees on its functionality while the recovery process is going on. We are especially interested in *searchability*, i.e., the functionality that search messages for a specific identifier are answered successfully if a node with that identifier exists in the network. We investigate overlay networks that are not only self-stabilizing but that also ensure that *monotonic* searchability is maintained while the recovery process is going on, as long as there are no corrupted messages in the system. More precisely, once a search message from node u to another node v is successfully delivered, all future search messages from u to v succeed as well. Monotonic searchability was recently introduced in OPODIS 2015, in which the authors provide a solution for a simple line topology. We present the first *universal* approach to maintain monotonic searchability that is applicable to a wide range of topologies. As the base for our approach, we introduce a set of primitives for manipulating overlay networks that allows us to maintain searchability and show how existing protocols can be transformed to use theses primitives. We complement this result with a generic search protocol that together with the use of our primitives guarantees monotonic searchability. As an additional feature, searching existing nodes with the generic search protocol is as fast as searching a node with any other fixed routing protocol once the topology has stabilized.

[*] This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center “On-The-Fly Computing” (SFB 901)

1 Introduction

In this paper, we continue our research started in [16] and investigate protocols for self-stabilizing overlay networks that guarantee the *monotonic* preservation of a characteristic that we call *searchability*, i.e., once a search message from node u to another node v is successfully delivered, all future search messages from u to v succeed as well. Instead of focusing on a specific topology, as done in [16], we present an approach that is aimed at universality. As a base, we present a set of primitives for overlay network maintainance for which we prove that they enable monotonic searchability. On top of that, we give a generic search protocol that, together with a protocol that solely uses these primitives, guarantees monotonic searchability. Additionally, we show that existing self-stabilizing overlay network protocols can be transformed to use our primitives.

To the best of our knowledge, we are the first to investigate monotonic searchability as an attempt to explore maintaining properties beyond the traditional "time and space" metrics during stabilization. We believe that the question of how to maintain monotonic searchability and similar properties during topological stabilization has a lot of potential for future research.

1.1 Model

We consider a distributed system consisting of a fixed set of nodes in which each node has a unique reference and a unique immutable numerical identifier (or short id). The system is controlled by a protocol that specifies the variables and actions that are available in each node. In addition to the protocol-based variables there is a system-based variable for each node called *channel* whose values are sets of messages. We denote the channel of a node u as $u.Ch$ and it contains all incoming messages to u. Its message capacity is unbounded and messages never get lost. A node can add a message to $u.Ch$ if it has a reference of u. Besides these channels there are no further communication means, so only point-to-point communication is possible.

There are two types of *actions* that a protocol can execute. The first type has the form of a standard procedure $\langle label\rangle(\langle parameters\rangle) : \langle command\rangle$, where *label* is the unique name of that action, *parameters* specifies the parameter list of the action, and *command* specifies the statements to be executed when calling that action. Such actions can be called locally (which causes their immediate execution) and remotely. In fact, we assume that every message must be of the form $\langle label\rangle(\langle parameters\rangle)$, where *label* specifies the action to be called in the receiving node and *parameters* contains the parameters to be passed to that action call. All other messages are ignored by nodes. The second type has the form $\langle label\rangle : \langle guard\rangle \longrightarrow \langle command\rangle$, where *label* and *command* are defined as above and *guard* is a predicate over local variables. We call an action whose guard is simply **true** a *timeout* action.

The *system state* is an assignment of values to every variable of each node and messages to each channel. An action in some node u is *enabled* in some system state if its guard evaluates to **true**, or if there is a message in $u.Ch$

requesting to call it. In the latter case, when the corresponding action is executed, the message is processed (and it is removed from $u.Ch$). An action is *disabled* otherwise. Receiving and processing a message is considered as an atomic step.

A *computation* is an infinite fair sequence of system states such that for each state S_i, the next state S_{i+1} is obtained by executing an action that is enabled in S_i. This disallows the overlap of action execution, i.e., action execution is *atomic.* We assume *weakly fair action execution* and *fair message receipt.* Weakly fair action execution means that if an action is enabled in all but finitely many states of a computation, then this action is executed infinitely often. Note that a timeout action of a node is executed infinitely often. Fair message receipt means that if a computation contains a state in which there is a message in a channel of a node that enables an action in that node, then that action is eventually executed with the parameters of that message, i.e., the message is eventually processed. Besides these fairness assumptions, we place no bounds on message propagation delay or relative nodes execution speeds, i.e., we allow fully asynchronous computations and non-FIFO message delivery. A *computation suffix* is a sequence of computation states past a particular state of this computation. In other words, the suffix of the computation is obtained by removing the initial state and finitely many subsequent states. Note that a computation suffix is also a computation. We say a state S' is reachable from a state S if starting in S there is a sequence of action executions such that we end up in state S'. We use the notion $S < S'$ as a shorthand to indicate that the state S happened chronologically before S'.

We consider protocols that do not manipulate the internals of node references. Specifically, a protocol is *compare-store-send* if the only operations that it executes on node references is comparing them, storing them in local memory and sending them in a message. In a compare-store-send protocol, a node may learn a new reference of a node only by receiving it in a message. A compare-store-send protocol cannot create new references. It can only operate on the references given to it.

The overlay network of a set of nodes is determined by their knowledge of each other. We say that there is a (directed) *edge* from a to b, denoted by (a, b), if node a stores a reference of b in its local memory or has a message in $a.Ch$ carrying the reference of b. In the former case, the edge is called *explicit*, and in the latter case, the edge is called *implicit.* Messages can only be sent via explicit edges. Note that message receipt converts an implicit edge to an explicit edge since the message is in the local memory of a node while it is processed. With NG we denote the directed *network (multi-)graph* given by the explicit and implicit edges. ENG is the subgraph of NG induced by only the explicit edges. A *weakly connected component* of a directed graph G is a subgraph of G of maximum size so that for any two nodes u and v in that subgraph there is a (not necessarily directed) path from u to v. Two nodes that are not in the same weakly connected component are *disconnected.* We assume that the positions of the processes in the topology are encapsulated in their identifier and that there is a distance measure which is based on the identifiers of the processes and which can be checked locally.

That is, for a given identifier ID, each node u can decide for each neighbor v whether v is closer to the node w with $id(w) = ID$ if such a node exists (we also say that $id(v)$ is closer to ID than $id(u)$ or $ds(id(v), ID) < ds(id(u), ID)$). For a node u, we define $R(u, ID)$ as the set containing u and all processes v for which there is a path Q from u to v via explicit edges such that for each edge (a, b) that is traversed in Q it holds that $ds(id(b), ID) < ds(id(a), ID)$. Furthermore, for a set U, we define $R(U, ID) := \bigcup_{u \in U} R(u, ID)$.

We are particularly concerned with search requests, i.e., $\textsc{Search}(v, destID)$ messages that are routed along ENG according to a given search protocol, where v is the sender of the message and $destID$ is the identifier of a node we are looking for. We assume that $\textsc{Search}()$ requests are initiated locally by an (possibly user controlled) application operating on top of the network. Note that $destID$ does not need to be an id of an existing node w, since it is also possible that we are searching for a node that is not in the system. If a $\textsc{Search}(v, destID)$ message reaches a node w with $id(w) = destID$, the search request *succeeds*; if the message reaches some node u with $id(u) \neq destID$ and cannot be forwarded anymore according to the given search protocol, the search request *fails*.

1.2 Problem Statement

A protocol is *self-stabilizing* if it satisfies the following two properties as long as no transient faults occur: (i) **Convergence:** starting from an arbitrary system state, the protocol is guaranteed to arrive at a legitimate state and (ii) **Closure:** starting from a legitimate state the protocol remains in legitimate states thereafter.

A self-stabilizing protocol is thus able to recover from transient faults regardless of their nature. Moreover, a self-stabilizing protocol does not have to be initialized as it eventually starts to behave correctly regardless of its initial state. In *topological self-stabilization* we allow self-stabilizing protocols to perform changes to the overlay network NG. A legitimate state may then include a particular graph topology or a family of graph topologies. We are interested in self-stabilizing protocols that stabilize to *static topologies*, i.e., in every computation of the protocol that starts from a legitimate state, ENG stays the same, as long as the node set stays the same.

In this paper we are not focusing on building a self-stabilizing protocol for a particular topology. Instead we are interested in providing a reliable protocol for searching in a wide range of topologies that fulfill certain requirements. Traditionally, search protocols for a given topology were only required to deliver the search messages reliably once a legitimate state has been reached. However, it is not possible to determine when a legitimate state has been reached. Furthermore, searching reliably during the stabilization phase is much more involved. We say a self-stabilizing protocol satisfies *monotonic searchability* according to some search protocol R if it holds for any pair of nodes v, w that once a $\textsc{Search}(v, id(w))$ request (that is routed according to R) initiated at time t succeeds, any $\textsc{Search}(v, id(w))$ request initiated at a time $t' > t$ will succeed. We do not mention R if it is clear from the context. A protocol is said to satisfy *non-trivial* monotonic searchability if (i) it satisfies monotonic searchability and (ii) every computation of the protocol contains a suffix such that for each pair of nodes v, w, SEARCH$(v, id(w))$ requests will succeed if there is a path from v to w in the target topology. Throughout the paper we will only investigate non-trivial monotonic searchability. Consequently, whenever we use the term monotonic searchability in the following, we implicitly refer to non-trivial monotonic searchability.

A *message invariant* is a predicate of the following form: If there is a message m in the incoming channel of a node, then a logical predicate P must hold. A protocol may specify one or more message invariants. An arbitrary message m in a system is called *corrupted* if the existence of m violates one or multiple message invariants. A state S is called *admissible* if there are no corrupted messages in S. We say a (self-stabilizing) protocol *admissibly satisfies* a predicate P if the following two conditions hold: (i) the predicate is satisfied in all computation suffixes of the protocol that start from admissible states, and (ii) every computation of the protocol contains at least one admissible state. A protocol *unconditionally satisfies* a predicate if it satisfies this predicate starting from any state.

The following was proven in [16]:

Lemma 1. *No self-stabilizing compare-store-send protocol can unconditionally satisfy monotonic searchability.*

Consequently, to prove monotonic searchability for a protocol (according to a given search protocol R) it is sufficient to show that: (i) in every computation of the protocol that starts from an admissible state, every state is admissible, (ii) in every computation of the protocol there is an admissible state, and (iii) the protocol satisfies monotonic searchability according to R in every computation that starts from an admissible state. Note that we have not defined any invariants yet and it is possible to pick invariants such that the set of admissible states equals the set of legitimate states, in which the problem becomes trivial. However, for the invariants we provide, any initial topology can be an admissible state. In particular, as long as no corrupt messages are initially in the system, our protocols satisfy monotonic searchability throughout the computation.

We will show that a broad class of existing self-stabilizing protocols can be transformed to satisfy monotonic searchability. More specifically, we will consider protocols that fulfill the MDL *property*, i.e., for any action a of the protocol it holds that (i) a node u executing action a will always keep a reference of another node v in its local memory if an edge (u, v) is part of the final topology, and (ii) if a node u executing action a in state S decides not to keep a reference of another node v in its local memory, every other action of the protocol executed by u in a subsequent state will decide to not keep the reference of v, and (iii) a node u executing action a decides *deterministically* and solely based on its *local* memory whether to send and where to send the reference of v, and (iv) in every legitimate state, for every reference of a node v contained in a message m in the channel of a node u (i.e., for any implicit edge (u, v)), there are fixed cycle-free paths $(u = u_1, u_2, \dots, u_k)$ such that u_i sends the reference of v to u_{i+1}, and u_k has an explicit edge (u_k, v) (note that there is only one path if the reference is never duplicated), i.e., the reference of v is forwarded along fixed paths until it finally fuses with an existing reference. Informally speaking, the first two properties imply that the protocol *monotonically* converges to its desired topology, since edges of the topology are always kept and edges that are not part of the topology are obviated over time. The last property implies that in legitimate states, all implicit edges will eventually merge with explicit edges. Note that the MDL property is generally not a severe restriction. Most existing protocols that stabilize to static topologies naturally fulfill this property.

In addition to the MDL property and the assumption that the final topology is static, we have one more condition on the topologies and their distance measures. The generic search protocol we will use to achieve monotonic searchability assumes that in the target topology for every pair of nodes u, v within the same connected component, there is a path of explicit edges from u to v with the property that each edge on the path strictly decreases the distance to v (i.e., for each edge (a, b) that is traversed in the path, $ds(id(b), id(v)) < ds(id(a), id(v))$). Note that many topologies naturally fulfill this property (in particular, whenever the distance is defined as the number of nodes on a shortest path).

1.3 Our contribution

To the best of our knowledge, we are the first to solve the problem of searching reliably during the stabilization phase in self-stabilizing topologies. Although routing with a low dilation is a major motivation behind the use of overlay topologies, prior to this work, one could not rely on the routing paths in such topologies[1]: In previous approaches, it can happen that a node u is able to send a message to a node v, while it is unable to do so in a later state, only because the system has not stabilized yet (which is not locally detectable by the nodes). In our solution, once a search message from a node u has successfully arrived at a node v, every further search message from u to v will also arrive at its destination, regardless of whether the system has fully stabilized or not.

We present a universal set of primitives for manipulating edges that protocols should use and a simple generic search protocol, which together satisfy monotonic searchability. Moreover, we provide a general description of how a broad class of self-stabilizing protocols for overlay networks can be transformed such that they use these primitives, thus satisfying monotonic searchability afterwards.

Our results of Section 3 may be of independent interest, where we reinvestigate the fundamental primitives for manipulating edges introduced in [13] and strengthen the results concerning the universality of these primitives.

2 Related work

The idea of self-stabilization in distributed computing was introduced by E.W. Dijkstra in 1974 [4], in which he investigated the problem of self-stabilization

[1] Note that [16] did solve the problem of monotonic searchability for the list, but the list has a worst-case routing time of $\Omega(n)$, thus not offering a low dilation.

in a token ring. In order to recover certain network topologies from any weakly connected state, researchers started with simple line and ring networks (e.g., [18, 7]). Over the years more and more topologies were considered, ranging from skip lists and skip graphs [14, 8], to expanders [6], and small-world graphs [12]. Also a universal algorithm for topological self-stabilization is known [1].

In the last 20 years many approaches have been investigated that focus on maintaining safety properties during convergence phase of self-stabilization, e.g. snap-stabilization [2, 3], super-stabilization [5], safe convergence [11] and self-stabilization with service guarantee [10]. Closest to our work is the notion of *monotonic convergence* by Yamauchi and Tixeuil [19]. A self-stabilizing protocol is monotonically converging if every change done by a node p makes the system approach a legitimate state and if every node changes its output only once. The authors investigate monotonically converging protocols for different classical distributed problems (e.g., leader election and vertex coloring) and focus on the amount of non-local information that is needed for them.

Research on monotonic searchability was initiated in [16], in which the authors proved that it is impossible to satisfy monotonic searchability if corrupted messages are present. In addition, they presented a self-stabilizing protocol for the line topology that is able to satisfy monotonic searchability.

3 Primitives for Topology Maintenance

An important property for every overlay management protocol is that weak connectivity is never lost by its own actions. Therefore, it is highly desirable that every node only executes actions that preserve weak connectivity. Koutsopoulos et al. [13] introduced the following four primitives for manipulating edges in an overlay network.

Introduction If a node u has a reference of two nodes v and w with $v \neq w$, u *introduces* w to v if u sends a message to v containing a reference of w while keeping the reference.

Delegation If a node u has a reference of two nodes v and w s.t. u, v, w are all different, then u *delegates* w's reference of v if u sends a message to v containing a reference of w and deletes the reference of w.

Fusion If a node u has two references v and w with $v = w$, then u *fuses* the two references if it only keeps one of these references.

Reversal If a node u has a reference of some other node v, then u *reverses* the connection if it sends a reference of itself to v and deletes its reference of v.

Note that the four primitives can be executed locally by every node in a wait-free fashion. Furthermore, for the Introduction primitive, it is possible that $w = u$, i.e., u introduces itself to v. The authors show that these four primitives are safe in a sense that they preserve weak connectivity (as long as there is no fault). This implies that *any* distributed protocol whose actions can be decomposed into these four primitives is guaranteed to preserve weak connectivity.

We define $\mathcal{IDF}$ as the set containing the first three primitives: Introduction, Delegation and Fusion. Let $\mathcal{P}_{\mathcal{IDF}}$ denote the set of all distributed protocols where all interactions between processes can be decomposed into the primitives of $\mathcal{IDF}$. According to [13] these protocols even preserve strong connectivity in a sense that for any pair of nodes u, v with a directed path in NG there will always be a directed path from u to v in NG. To the best of our knowledge, all self-stabilizing topology maintenance protocols proposed so far (such as the list [18, 15, 7], the Delaunay graph [9], etc.) satisfy this property. Moreover, in [13], the four primitives were shown to be *universal*, i.e. the primitives allow one to get from any weakly connected graph $G = (V, E)$ to any other weakly connected graph $G' = (V, E')$ for NG. In fact, only the first three primitives (i.e., $\mathcal{IDF}$) are necessary to get from any weakly connected graph to any *strongly* connected graph, which is sufficient in our case ([13] denote this by *weak universality*). Note that the notion of universality for a set of primitives is not constructive, i.e., only *in principle* the primitives allow one to get from any weakly connected graph to any other weakly connected graph. We strengthen the results concerning universality of the primitives with the following theorem (the proof can be found in the full version [17]).

Theorem 1. *Any compare-store-send protocol that self-stabilizes to a static strongly-connected topology and preserves weak connectivity can be transformed such that the interactions between nodes can be decomposed into the primitives of* $\mathcal{IDF}$.

4 Primitives for Monotonic Searchability

Although the primitives of [13] are general enough to construct any conceivable overlay, they do not inherently satisfy monotonic searchability. This is due to the fact that the Delegation primitive replaces an explicit edge (u, v) by a path (u, w, v) consisting of an explicit edge (u, w) and an implicit edge (w, v) and thus a search message from u to v issued after the delegation may be processed by w before there is a path from w to v via explicit edges, causing the search message to fail (even though an earlier message sent while (u, v) was still an explicit edge was delivered successfully). Consequently, we are going to introduce a new set of primitives that enables monotonic searchability. We say a set of primitives is *search-universal* according to a set of Invariants $\mathcal{I}$ if the following holds:

1. the set of primitives is weakly universal,
2. starting from every state in which the invariants in $\mathcal{I}$ hold, for every pair of nodes u and v as soon as there is a path via explicit edges from u to v, there will be a path via explicit edges from u to v in every subsequent step.

We are now going to introduce a modified set of primitives that are search-universal. Moreover, we will show that these new primitives are also general enough to cover all self-stabilizing protocols that can be built by the original primitives. Consequently, we ultimately aim at a result similar to Theorem 1 for the new primitives.

Remember that we assume the MDL property. Therefore, in every fixed state S in every execution of a self-stabilizing protocol, each node u can divide its explicit edges into two subsets: the *stable edges* and the *temporary edges* (not to be confused with implicit edges). The first set contains those explicit edges that u wants to keep, given its current neighborhood in S; the second set holds the explicit edges that are not needed from the perspective of u in S. Note that the set of temporary edges can also be the empty set.

For the new primitives, a node does not only store references of its neighbors, but additionally stores sequence numbers for every reference in its local memory, i.e., every node u stores for each neighbor v an entry $u.eseq[id(v)]$ (or $u.eseq[v]$, in short). We keep the Introduction primitive as in Section 3 and change Delegation and Fusion in the following way:

Safe-Delegation Consider a node u that has references of two different nodes v and w. In order to perform *Safe-Delegation*, u has to distinguish between (u, w) being implicit or temporary.
If (u, w) is an implicit edge, it is delegated as in the original delegation primitive (we will later refer to this case as an *implicit delegation* or IMPLDELEGATE() to avoid confusion with the original primitives). If (u, w) is a temporary edge, it can only be delegated to a node v if (u, v) is a stable edge. Whenever an explicit edge (u, w) is to be delegated to another node v, u sends a DELEGATEREQ$(u, w, eseq)$ message to v, where $eseq = u.eseq[w]$. Additionally, it sets $u.eseq[v]$ to $max\{u.eseq[v], u.eseq[w]+1\}$. Any node v that receives a DELEGATEREQ$(u, w, eseq)$ message, adds (v, w) to its set of explicit edges (if it does not already exist), sets $v.eseq[w]$ to $\max\{v.eseq[w], eseq+1\}$ and sends a DELEGATEACK$(w, eseq)$ message back to u. Upon receipt of this message, u checks whether $eseq = u.eseq[w]$ and whether (u, w) is actually a temporary edge (note that the last check is necessary to handle corrupt initial states). If both conditions hold, u removes the temporary explicit edge to w and sends an IMPLDELEGATE(w) message to one of its neighbors. Otherwise, u simply acts as it would upon receipt of an IMPLDELEGATE(w) message.

Fusion If a node u has two references v and w with $v = w$, then u *fuses* the two references if it only keeps one of these references. Note that when a node u receives a DELEGATEREQ(v,w,eseq) message and already stores a reference of w, it also behaves as described in the Safe-Delegation primitive.

We define $\mathcal{ISF}$ as the set containing the three primitives Introduction, Safe-Delegation and Fusion. Throughout the paper we assume that DELEGATEREQ() and DELEGATEACK() messages are only sent in the Safe-Delegation primitive. Analogous to $\mathcal{P}_{\mathcal{IDF}}$, let $\mathcal{P}_{\mathcal{ISF}}$ denote the set of all distributed protocols where all interactions between processes can be decomposed into the primitives of $\mathcal{ISF}$. Likewise to the MDL property, we say that a protocol fulfills the *stable* MDL property, if the protocol fulfills the MDL property with respect to stable explicit edges. More specifically, for any action a of the protocol it holds that a node u executing action a will always keep a reference of another node v in its local memory (i.e., the stable edge (u, v)) if an edge (u, v) is part of the final topology, and if a node u executing action a in some state S decides to not keep a reference of another node v in its local memory (i.e., the temporary or implicit edge (u, v)), every other action of the protocol executed by u in any state $S' > S$ will decide to not keep the reference of v.

4.1 Universality of the new primitives

To show that our primitives are search-universal we first show that they are weakly universal. The corresponding proof can be found in the full version.

Lemma 2. *$\mathcal{ISF}$ is weakly universal.*

In order to enable monotonic searchability, we define the following two **message invariants**:

1. If there is a DELEGATEREQ(u,w,eseq) message in $v.Ch$, then there exists a path $P = (u = x_1, x_2, \dots, x_k = v)$ that does not contain (u, w) and for every $1 \le i < k$, $x_i.eseq[x_{i+1}] > u.eseq[w]$, or $u.eseq[w] > eseq$.
2. If there is a DELEGATEACK(w,eseq) message in $u.Ch$, then there exists a path $P = (u = x_1, x_2, \dots, x_k = w)$ that does not contain (u, w) and for every $1 \le i < k$, $x_i.eseq[x_{i+1}] > u.eseq[w]$, or $u.eseq[w] > eseq$.

Intuitively, Invariant 1 says that whenever node v has a DELEGATEREQ(u,w,eseq) message in $v.Ch$ (i.e., node u asked v to establish the edge (v, w) such that it may remove its own (u, w) edge), then there is a path from u to v that does not use the edge (u, w). Invariant 2 states that whenever a node u has a DELEGATEACK(w,seq) message in $u.Ch$ (i.e., some other node v which u asked to establish the edge (v, w) has already done so), then there is a path from u to w that does not use the edge (u, w). However, both statements only need to hold if the value of ESEQ indicates that the messages belong to a current safe-delegation, i.e., if $u.seq[w] > eseq$, the DELEGATEREQ() or DELEGATEACK() message can be ignored.

We define the predicate $E(u, v)$ to be true if and only if there exists a directed path from u to v via explicit edges. In order to show search-universality, we prove the following lemma.

Lemma 3. *Consider a computation of a protocol $P \in \mathcal{P}_{\mathcal{ISF}}$ that fulfills the stable* MDL *property. If there is a state S such that Invariant 1 and Invariant 2 hold, then they will hold in every subsequent state. Additionally, for every state $S' \ge S$ it holds that if $E(u, v) \equiv TRUE$ in S', then $E(u, v) \equiv TRUE$ in every state $S'' \ge S'$.*

Lemma 2 and Lemma 3 imply the following corollary:

Corollary 1. *$\mathcal{ISF}$ is search-universal according to Invariant 1 and Invariant 2.*

We conclude this section by showing that a protocol $A \in \mathcal{P}_{\mathcal{IDF}}$ that fulfills the MDL property and self-stabilizes to some topology can be transformed into a protocol $B \in \mathcal{P}_{\mathcal{ISF}}$ that fulfills the stable MDL property and for which it holds that in every computation of B there is a state in which Invariant 1-2 hold. The corresponding proof can be found in the full version [17].

Theorem 2. *Consider a protocol $A \in \mathcal{P}_{\mathcal{IDF}}$ that self-stabilizes to a strongly-connected topology T and that fulfills the* MDL *property. Then A can be transformed into another protocol $B \in \mathcal{P}_{\mathcal{ISF}}$ such that B fulfills the stable* MDL *property, B self-stabilizes to the same topology, and in every computation of B there exists a computation suffix in which Invariants 1 and 2 hold.*

5 The generic search protocol

In this section we describe a generic search protocol such that every protocol in $\mathcal{P}_{\mathcal{ISF}}$ fulfilling the stable MDL property satisfies monotonic searchability according to that search protocol. We assume that when a node u wants to search for a node with identifier ID, it performs an InitiateNewSearch(ID) action in which a Search(u, ID) message is created. The search request is regarded as answered as soon as the Search(u, ID) message is either dropped, i.e., it *fails*, or is received by the node w with $id(w) = ID$, i.e. it *succeeds*.

The principle idea of the *generic search protocol* is the following: A node u with a Search(u, ID) message does not directly forward this message through the network but buffers it. Instead, u initiates a probing algorithm whose goal is to either receive the reference of the node w with $id(w) = ID$, or to get a negative response in case this node does not exist or cannot be reached yet. In the former case, u directly sends Search(u, ID) to w. In the latter case, u drops Search(u, ID). Whenever an additional Search(u, ID) message for the same identifier ID is initiated at u while a probing for ID is still in progress, this message is combined with previous Search(u, ID) messages waiting at u.

For the probing, a node u with a buffered Search(u, ID) message periodically initiates a new Probe() message in its Timeout action. This Probe() message contains four arguments: First, a reference *source* of the source of the Probe() message., i.e., a reference of u. Second, the identifier *destID* of the node that is searched, i.e., ID. Third, a set $Next$ that holds references of all neighbors of u with a closer distance to *destID* than $id(u)$. Last, a sequence number *seq* that is used to distinguish probe messages that belong to different probing processes from the same node and for the same target, i.e., $seq = u.seq[ID]$, where $u.seq[ID]$ is a value stored at u. This is necessary because in each execution of the Timeout action, a new probe message is sent, although upon receival of the first response to such a message, the set of buffered search messages is sent out to the target or dropped completely. Thus, future replies may arrive afterwards and u has to know that these are outdated. All in all, u initiates a Probe($source, destID, Next, seq$) message and sends this message to the node in $Next$ whose identifier has the maximum distance to ID (i.e., it is the closest to u).

Any intermediate node v that receives a Probe($source, destID, Next, seq$) message first checks whether $id(v) = destID$. If so, v sends a reference of itself to *source* via a ProbeSuccess($destID, dest$) message with $dest = v$. Otherwise, v removes itself from $Next$ and adds all its neighbors to $Next$ that have a closer distance to *destID* than itself. If $Next$ is empty after this step, v responds to *source* via a ProbeFail($destID, seq$) message. Otherwise, v

forwards the $\textsc{Probe}(\mathit{source}, \mathit{destID}, \mathit{Next}, \mathit{seq})$ message (with the already described changes performed to Next) to the node in Next whose identifier has the maximum distance to ID. If the initiator u of a probe receives a $\textsc{ProbeSuccess}(\mathit{destID}, \mathit{dest})$ or a $\textsc{ProbeFail}(\mathit{destID}, \mathit{seq})$ message, it first checks whether $\mathit{seq} \geq u.\mathit{seq}[\mathit{destID}]$, i.e., it checks whether the received message is a response to the current batch of search requests. If it is from an earlier probe, u simply drops the received message. Otherwise, u acts depending on the message it received: In case of a $\textsc{ProbeSuccess}(\mathit{destID}, \mathit{dest})$ message, u sends out all (possibly combined) $\textsc{Search}(u, \mathit{destID})$ messages waiting at u to dest (thus stopping the probing). In case of a $\textsc{ProbeFail}(\mathit{destID}, \mathit{seq})$ message, u drops all $\textsc{Search}(u, \mathit{destID})$ messages waiting at u to dest (thus also stopping the probing). In both cases, u additionally increases $u.\mathit{seq}[\mathit{destID}]$ such that probe messages that are still in the system at this point in time cannot have any effects on future requests. The pseudocode of the generic search protocol and supplementary details can be found in the full version [17].

Using the protocol as specified above could cause a high dilation because each probe message in each step is always sent to the node with the highest distance to the target in Next, even if a shorter path is possible. Luckily, if there exists a fast routing protocol for the stabilized target topology (i.e., $o(n)$ hops in the worst case), it is possible to speed up search messages in legitimate states (and possibly even earlier). Details can be found in the full version [17].

As the generic search protocol cannot guarantee to function properly under the presence of corrupt messages, we define the following additional invariants that are maintained during the execution of the generic search protocol (that did not start with corrupt messages):

3. If there is a $\textsc{Probe}(\mathit{source}, \mathit{destID}, \mathit{Next}, \mathit{seq})$ message in $u.\mathit{Ch}$, then
   (a) $u \in \mathit{Next}$ and $\forall w \in \mathit{Next} \setminus \{u\} : \mathit{ds}(\mathit{id}(w), \mathit{destID}) \leq \mathit{ds}(\mathit{id}(u), \mathit{destID})$,
   (b) $R(\mathit{Next}, ID) \subseteq R(\mathit{source}, ID)$, and
   (c) if v exists with $\mathit{id}(v) = \mathit{destID}$ and $v \notin R(\mathit{Next}, \mathit{destID})$, then for every admissible state with $\mathit{source}.\mathit{seq}[\mathit{destID}] < \mathit{seq}$, $v \notin R(\mathit{source}, \mathit{destID})$.
   If there is a $\textsc{FastProbe}(\mathit{source}, \mathit{destID})$ message in $u.\mathit{Ch}$, then
   (d) $u \in R(\mathit{source}, \mathit{destID})$.
4. If there is a $\textsc{ProbeSuccess}(\mathit{destID}, \mathit{dest})$ message in $u.\mathit{Ch}$, then $\mathit{id}(\mathit{dest}) = \mathit{destID}$ and $\mathit{dest} \in R(u, \mathit{destID})$.
5. If there is a $\textsc{ProbeFail}(\mathit{destID}, \mathit{seq})$ message in $u.\mathit{Ch}$, then if v exists such that $\mathit{id}(v) = \mathit{destID}$, then for every admissible state with $u.\mathit{seq}[\mathit{destID}] < \mathit{seq}$, $v \notin R(u, \mathit{destID})$.
6. If there is a $\textsc{Search}(v, \mathit{destID})$ message in $u.\mathit{Ch}$, then $\mathit{id}(u) = \mathit{destID}$ and $u \in R(v, \mathit{destID})$.

We say a protocol for the self-stabilization of a topology is *monotonic-searchability-sufficient* (*ms-sufficient*) if (i) all interactions between processes can be decomposed into the primitives in $\mathcal{ISF}$, (ii) it fulfills the stable MDL property, (iii) it uses the generic search protocol for searching, (iv) no $\textsc{Probe}()$, $\textsc{ProbeSuccess}()$, $\textsc{ProbeFail}()$, or $\textsc{Search}()$ message is sent at any other occasion than the ones specified in the generic search protocol, and (v) in every computation of the protocol there is a state in which the first two invariants hold. Note that Theorem 2 implies the following:

Corollary 2. *Any conventional protocol $A \in \mathcal{P}_{\mathcal{IDF}}$ that self-stabilizes to a strongly-connected topology T and that fulfills the* MDL *property can be transformed into an ms-sufficient protocol that stabilizes the same topology.*

For an ms-sufficient protocol, we define a state as admissible if all six invariants hold. The proof of the following theorem can be found in the full version [17].

Theorem 3. *Every ms-sufficient protocol satisfies monotonic searchability according to Invariant 1-6.*

The following result follows from the description of the generic search protocol:

Corollary 3. *Every ms-sufficient protocol P that stabilizes to a topology T and in which the generic search protocol uses a routing strategy with a worst-case routing time of $O(T(n))$ for the fast search as described in the protocol, then P answers successful search requests in legitimate states in time $O(T(n))$.*

6 Conclusion and Outlook

In this work we further strengthened the notion of monotonic searchability introduced in [16] by presenting a universal approach for adapting conventional protocols for topological self-stabilization such that they satisfy monotonic searchability. Even more, we carved out some design principles that protocols should adhere to in order to enable reliable searches even during the stabilization phase.

Although our results solve the problem of monotonic searchability for a wide range of topologies, there are certain aspects that have not been studied yet. For example, we did not consider the additional cost of convergence (i.e., the amount of additional messages to be sent), nor the impact of our methods on the convergence time of the topology. Additionally, while our generic search protocol enables us to search existing nodes in legitimate states with a low dilation, searching for a non-existing node can still cause a message to travel $\Omega(n)$ hops, even in a legitimate state. Whether this is provably necessary or could be improved is still an open question.

References

1. Andrew Berns, Sukumar Ghosh, and Sriram V. Pemmaraju. Building self-stabilizing overlay networks with the transitive closure framework. *Theor. Comput. Sci.*, 512:2–14, 2013.
2. Alain Bui, Ajoy Kumar Datta, Franck Petit, and Vincent Villain. Snap-stabilization and PIF in tree networks. *Distributed Computing*, 20(1):3–19, 2007.
3. Sylvie Delaët, Stéphane Devismes, Mikhail Nesterenko, and Sébastien Tixeuil. Snap-stabilization in message-passing systems. *J. Parallel Distrib. Comput.*, 70(12):1220–1230, 2010.

4. Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
5. Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago J. Theor. Comput. Sci.*, 1997, 1997.
6. Shlomi Dolev and Nir Tzachar. Spanders: Distributed spanning expanders. *Sci. Comput. Program.*, 78(5):544–555, 2013.
7. Dominik Gall, Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. A note on the parallel runtime of self-stabilizing graph linearization. *Theory Comput. Syst.*, 55(1):110–135, 2014.
8. Riko Jacob, Andréa W. Richa, Christian Scheideler, Stefan Schmid, and Hanjo Täubig. Skip$^+$: A self-stabilizing skip graph. *J. ACM*, 61(6):36:1–36:26, 2014.
9. Riko Jacob, Stephan Ritscher, Christian Scheideler, and Stefan Schmid. Towards higher-dimensional topological self-stabilization: A distributed algorithm for delaunay graphs. *Theor. Comput. Sci.*, 457:137–148, 2012.
10. Colette Johnen and Fouzi Mekhaldi. Robust self-stabilizing construction of bounded size weight-based clusters. In *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part I*, pages 535–546, 2010.
11. Hirotsugu Kakugawa and Toshimitsu Masuzawa. A self-stabilizing minimal dominating set algorithm with safe convergence. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Proceedings, 25-29 April 2006, Rhodes Island, Greece*, 2006.
12. Sebastian Kniesburges, Andreas Koutsopoulos, and Christian Scheideler. A self-stabilization process for small-world networks. In *26th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 1261–1271, 2012.
13. Andreas Koutsopoulos, Christian Scheideler, and Thim Strothmann. Towards a universal approach for the finite departure problem in overlay networks. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*, pages 201–216, 2015.
14. Rizal Mohd Nor, Mikhail Nesterenko, and Christian Scheideler. Corona: A stabilizing deterministic message-passing skip list. *Theor. Comput. Sci.*, 512:119–129, 2013.
15. Melih Onus, Andréa W. Richa, and Christian Scheideler. Linearization: Locally self-stabilizing sorting in graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*, 2007.
16. Christian Scheideler, Alexander Setzer, and Thim Strothmann. Towards establishing monotonic searchability in self-stabilizing data structures. In *Principles of Distributed Systems - 19th International Conference, OPODIS 2015, Proceedings*, 2015.
17. Christian Scheideler, Alexander Setzer, and Thim Strothmann. Towards a universal approach for monotonic searchability in self-stabilizing overlay networks (full version). *ArXiv e-prints*, July 2016.
18. Ayman Shaker and Douglas S. Reeves. Self-stabilizing structured ring topology P2P systems. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005), 31 August - 2 September 2005, Konstanz, Germany*, pages 39–46, 2005.
19. Yukiko Yamauchi and Sébastien Tixeuil. Monotonic stabilization. In *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*, pages 475–490, 2010.