Studying Network Optimization in the Context of Self-Stabilization

Authors: Abusayeed Saifullah

Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A self-stabilizing system is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a legitimate global state, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.

... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

https://openscholarship.wustl.edu/cse_research/88
Studying Network Optimization in the Context of Self-Stabilization

Complete Abstract:

Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A self-stabilizing system is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a legitimate global state, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/88
STUDYING NETWORK OPTIMIZATION IN THE CONTEXT OF SELF-STABILIZATION

Abusayeed Saifullah
Department of Computer Science and Engineering
Washington University in St Louis
saifullaha@cse.wustl.edu

ABSTRACT
Self-stabilization is a theoretical framework of non-masking fault-tolerance for distributed networks. A self-stabilizing system is capable of tolerating any unexpected transient fault without outside intervention and, regardless of the initial state, it can converge to a legitimate global state, a predefined vector of local states, in finite time. Self-stabilization has rendered a good problem solving paradigm of networks over the last decade. In this paper, we survey the self-stabilizing solutions for various network optimization problems such as network flow, load balancing, load and resource distribution, routing, file distribution, shortest paths etc. The paper also summarizes some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems. The works pertaining to gradient adaptation of self-stabilizing system are also presented.

1 Introduction
The notion of self-stabilization was introduced by Dijkstra in 1974 in his classic paper [47]. Dijkstra [47] studies the scenario when there is no global memory and, therefore, the current global state of the system is recorded in variables distributed over the various processes where the communication facilities are restricted only between neighbors. Based on the network topology and signal propagation delay, each node gets only a partial view of the global state. Dijkstra [47] notices the complication that the behavior of a process can only be influenced by that part of the total current system state description that is available to it; local actions taken on account of local information must accomplish a global objective.

Uncertain as to whether the local moves can assure convergence towards satisfaction of such a global criterion, Dijkstra [47] limited his attention to a ring of finite-state machines and provided its solution for self-stabilization which he proved later in [48]. This was the idea that did break the ice to encompass a formal and unified approach to fault tolerance. For almost one decade following the inception, this brilliant work was not noticed and very few papers were published in this area. Once this proposal was recognized as a milestone in works on fault tolerance, the notion propagated among the distributed system community rapidly that resulted in a burst of papers [128, 125, 126, 127, 129] as well as workshops and symposiums devoted entirely
to this area.

The adoption of self-stabilization is of great importance where a system is vulnerable to transient faults. In modern distributed networks, various transient faults are likely to occur as these systems are exposed to constant changes of their environment. Since most self-stabilizing algorithms are non-terminating, if the system experiences transient faults corrupting the local states of the computing elements, once faults cease, the algorithms themselves guarantee to recover in finite time to a safe state without being assisted by any outside agent. This also means that the complicated task of initializing the system is no longer needed, since the algorithms regain correct behavior regardless of initial state. Self-stabilization provides a formal and unified approach to fault tolerance with respect to a model of transient failures and makes the departure from previous approaches to fault tolerance in distributed network.

At present, self-stabilization has rendered a good problem solving paradigm for distributed networks. Many network optimization problems such as network flow, load balancing, routing, load and resource distribution, file distribution, shortest paths etc. have been solved in the context of self-stabilization. In this paper, we survey the self-stabilizing solutions of network optimization problems available in the literature (Section 3 to Section 13). Some recent works presenting how the convergence of a self-stabilizing distributed network can be modelled as a convex optimization problem with the exploitation of an analogy between self-stabilizing systems and stable feedback systems are also presented (Section 3). We also summarize the works pertaining to gradient adaptation of self-stabilizing systems (Section 6). Finally, we propose some related promising open problems along with some concluding remarks in Section 14.

2 Preliminaries

Some concepts and definitions related to this survey are in order and are explained as follows:

2.1 Distributed Network

A distributed system is generally defined as a set of processing elements or state machines interconnected by a network of some fixed topology. The types of hardware, operating systems, and other resources may vary drastically. It is similar to computer clustering with the main difference being a wide geographic dispersion of the resources. In the shared register or local shared memory model of distributed network, communication is restricted only between neighbors. That is, there is no common or global memory. Each processor can communicate only with the neighbors and the communication is carried out by using a shared communication channel or link. Directly connected processors are called each other’s neighbors. By contrast, in a message passing model of distributed system, communications are carried out by sending and receiving messages.

From the local and global perspective two types of states are defined:

Local State: Each node maintains a set of local variables whose contents specify the local state of that node.
Global State: This is the state of the system as a whole. The global state is expressed in terms of the local states of the processors. Specifically, the global state is a vector $c \in (s_1 \times s_2 \times \cdots \times s_n)$, where $s_i$ is the local state of machine $i$.

2.2 Fault Tolerance

Implicit in the definition of fault tolerance is the assumption that there is a specification of what constitutes the correct state and behavior of the system. A failure occurs when an actual running system deviates from this specification [130]. The cause of a failure is called an error. An error represents an invalid system state, one that is not allowed by the system behavior specification. The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure [130].

Most of the phenomena that contribute to the unexpected perturbation of the system state are indistinguishable. Some of them, explained in [130] and [129], are:

- **Processing Machine Failures**
- **Inconsistent Initialization**
- **Mode Change**
- **Communication Media Failures**
- **Transmission Delays**
- **Distributed Agreement Problems**

Based on duration, faults can be classified as transient or permanent [130]. A transient fault is an event that may change the state of a system by corrupting the local states of the machines. A particularly problematic type of transient fault is the intermittent fault that recurs, often unpredictably [130].

Fault tolerance is the ability of a system to perform its function correctly even in the presence of internal faults. Fault-tolerant computing is extremely hard since it involves intricate algorithms to cope with the inherent complexity of the physical world. As it turns out that the world conspires against us and is constructed in such a way that, generally, it is simply not possible to devise an absolutely foolproof or hundred percent reliable system [60]. No matter how hard we try, there is always a possibility that something can go wrong. The best we can do is to reduce the probability of failure to an acceptable level. Unfortunately, the more we strive to reduce this probability, the higher becomes the cost.

2.3 Self-Stabilizing System

The property of self-stabilization can recover the system from transient faults and represents a departure from previous approaches to fault tolerance. The global correctness criterion of the system is defined by a predicate which is a boolean function over the whole system. Based on this predefined function, two classes of global states are defined:

- **Legitimate State:** When the system satisfies the predicate it is said that the system is in a legitimate state.
- **Illegitimate State:** The state of the system is said illegitimate when it fails to satisfy the predicate.

The goal of a self-stabilizing algorithm is to start from an arbitrary (possibly illegitimate) state and then to
reach a legitimate state after a finite number of steps (moves). For each machine one or more privileges are defined. A privilege (also known as guard) of a processor is a boolean function of its own state and the states of its neighbors. A processor that satisfies its privilege can make a move. A move is an action taken by a processor that changes the local state of the processor. Therefore, each move takes the system into a new global state.

In addition to this predicate of global correctness criterion, Dijkstra [47] defines the following four constraints of a legitimate state of the system:

1. In each legitimate state one or more privileges will be present.

2. In each legitimate state, each possible move will bring the system again in another legitimate state.

3. Every privilege exists in at least one legitimate state.

4. For any pair of legitimate states, there exists a sequence of moves that can transfer the system from one state to the other.

However, depending on the system specifications and criteria of the problems, these requirements have been modified in some papers. Schneider [129] introduces a generalization of self-stabilization based on Arora and Gouda [11] and Arora [10]. He defines a system X as self-stabilizing with respect to a predicate P if X satisfies the following two properties:

1. Closure: P is closed under the execution of X. That is, once P is established in X, it cannot be falsified.

2. Convergence: Starting from an arbitrary global state, X is guaranteed to reach a global state satisfying P within a finite number of state transitions (moves).

A self-stabilizing algorithm can be encoded as a set of rules. Each rule has two parts: the privilege and the move as shown below:

\[
\text{if } <\text{privilege}> \text{ then } <\text{move}>
\]

If two or more processors satisfy their privileges at the same time, only one of them is arbitrarily selected by a daemon (central or distributed). Two variations of self-stabilization, called super-stabilization and silent stabilization, are proposed in [55] and [50], respectively.

3 Convergence Analysis

Since verification of the convergence (to predicate or correct global state) criterion is tedious, the vision is to provide for proofs of self-stabilization in an automatic fashion. One excellent but difficult strategy to prove the correctness is to use a bounded function defined over the entire system. The bounded function has been used in [62, 90, 35, 91, 93] whose value monotonically decreases with computing steps. However, in [6, 134, 8], no bounded function has been used. Correctness can also be proven by mathematical induction [8, 9] and graph theoretical reasoning [134, 6].

Oehlerking et al. [116] study the convergence to the global correctness criterion from a very different perspective. The paper points out the analogy between self-stabilizing systems and stable feedback systems and thus self-stabilization and Lyapunov stabilization are treated exactly alike. A continuous-time linear
time-invariant system $S$ is said to be Lyapunov stable if and only if all the eigenvalues of $S$ have real parts less than or equal to 0, and those with real parts equal to 0 are nonrepeated. When the system behaves like a piecewise-affine hybrid system, i.e.,

$$x[k+1] = A_m x + b_m, A_m \in \mathbb{R}^{n \times n}, b_m \in \mathbb{R}^n,$$

where $x$ is the vector of states of all machines of the system, then the convergence to the predicate is simply the well-known convex optimization \[27\]. Let

$$x[k+1] = f(x[k])$$

(since the state change is the function of previous state) be a discrete-time system with $f(0) = 0$. If $\exists V : \mathbb{R}^n \to \mathbb{R}$ such that $V(x)$ is positive definite, $\dot{V}(x) = V(f(x)) - V(x)$ is negative definite, and $V(x) \to \infty$ for $\|x\| \to \infty$, then $V$ is a Lyapunov function and the system is globally asymptotically stable in $0$. Oehlerking et al. \[116\] use quadratic Lyapunov function candidates and reduce the problem of finding a Lyapunov function to convex optimization via so-called linear matrix inequalities problem:

Find $k_i \in \mathbb{R}$ such that

$$x^T (F_0 + \sum_{i=1}^N k_i F_i) x \geq 0, \forall x \in \mathbb{R}^n,$$

where $F_j \in \mathbb{R}^{n \times n}$, $0 \leq j \leq N$, are fixed.

Herman \[85\] and Beauquier et al. \[22\] explain convergence from the probabilistic point of view. In \[85\], for token ring circulation, the convergence properties are interpreted in terms of a function $D$ that partitions the set of ring states with at least two tokens, where $D.x$ is the minimum distance between two tokens in ring state $x$. The expected value of $k$ such that $D.(f^k.x) = 1$ is:

$$E[kC.k] = e_j (I - A)^{-1} A (I - A)^{-1} e_1^T$$

(2)

The expected time for convergence follows from (2):

$$E[1 + kC.k/4] = 1 + E[kC.k]/4 = n^2/2.$$

Kakugawa \[102\] introduces a description language and its processor for self-stabilizing systems of arbitrary network topology to verify mechanically. For an incorrect self-stabilizing system, the verification system outputs a counterexample which consists of an initial system state and an execution sequence which does not converge.

## 4 Time and Space Optimality

Great efforts have been given to achieve time, space, and state optimality \[28\] in the adoption of self-stabilization towards solving different problems in recent years. Aggarwal \[3\] is the first self-stabilizing algorithm to compute the spanning tree of an asynchronous network which is time-optimal (i.e. stabilizes in time $O(\text{diameter})$) without any prior knowledge of the network size. It presents both a randomized Las-Veglas algorithm for anonymous network and a deterministic version of ID-based network. In \[49\], time optimal self-stabilizing dynamic protocols for a variety of tasks including routing, leader election, topology update are available. Each of this protocols sta-
bilizes in $O(d)$ time, where $d$ is the diameter of the system.

Gradinariu and Johnen [81] propose a self-stabilizing probabilistic solution for the neighborhood unique naming problem in uniform, anonymous networks with arbitrary topology. Their solution is time optimal i.e. naming is done in only one trial per processor in the average under any unfair scheduler. Efficiency and simplicity for self-stabilizing token circulation in trees and arbitrary networks have been extensively studied by Petit [120]. For different network topology the service time and space of this protocol have been analyzed in [95, 22, 96, 19]. Space optimality for stabilizing leader election is achieved in the randomized algorithm of Beauquir et al. [20]. Awerbuch [15] considers the question of fault-tolerant distributed network protocols with extremely small memory requirements per processor. Specifically, he shows that even in the case of worst-case transient faults, many fundamental network protocols can be achieved using only $O(\log n)$ bits of memory per incident network edge. A time-optimal self-stabilizing network synchronization algorithm is provided in [14].

Ghodsi [67] proposes a self-stabilizing network size estimation gossip algorithm for peer to peer network. Memory requirements for several silent stabilizing algorithms have been analyzed by Dolev [50]. Dolev [50] shows that any center-finding, leader election or tree construction algorithm of this class require $\Omega(\log n)$ bits per communication register or process.

5 Optimized Network Flow

A combination of the optimized network flow algorithms of Ford-Fulkerson [63] and Edmonds-Karp [59] has been implemented in the self-stabilizing model by Ghosh et al. [72]. The algorithm uses an approach similar to that of Goldberg and Tarjan [79]. For each edge $(i, j)$, there is a register or variable $f(i, j)$ at both node $i$ and node $j$. At every node $i$, the variable $d(i)$ contains the believed shortest path from $s$ (source) to $i$. For every node $i \neq s, t$, $demand(i) = O_f(i) - I_f(i)$, where $O_f(i)$ and $I_f(i)$, at $i$, are outflow and inflow, respectively (and $t$ is the sink). Each node $i$, tries to restore flow conservation constraint, $demand(i) = 0$, either by reducing inflow or increasing outflow if $demand(i) < 0$, or by increasing inflow or reducing outflow if $demand(i) > 0$. Each node with positive demand attempts to pull flow via a shortest path from $s$ to itself in the residual graph. If the node believes that a path from $s$ to itself does not exist in the residual graph, then it rejects the demand by pushing it back along an outgoing edge.

Four types of privileges or guarded commands are:\footnote{$\text{IN}(i)$ : incoming edges of $i$; $C(i, j)$ : capacity of $(i, j)$; $r(i, j) = C(i, j) - f(i, j)$; $Reduce_{\text{Inflow}}(i)$; $Reduce_{\text{Outflow}}(i)$; $S_1 : d(i) \neq \min\{d(p) + 1|p \in \text{IN}(i)\} \cup \{n\} \Rightarrow d(i) := \min\{d(p) + 1|p \in \text{IN}(i)\} \cup \{n\}$; $S_2 : demand(i) < 0 \Rightarrow Reduce_{\text{Inflow}}(i)$; $S_3 : (\exists j \in \text{IN}(i) : demand(i) > 0 \land d(i) < n \land d(j) = d(i) - 1) \Rightarrow (f(j, i) := f(j, i) + min(demand(i), r(j, i)))$; $S_4 : demand(i) > 0 \land d(i) = n \land i \neq t \Rightarrow Reduce_{\text{Outflow}}(i)$;}

\begin{align*}
S_1 : d(i) &\neq \min\{d(p) + 1|p \in \text{IN}(i)\} \cup \{n\} \Rightarrow d(i) := \min\{d(p) + 1|p \in \text{IN}(i)\} \cup \{n\}; \\
S_2 : demand(i) < 0 &\Rightarrow \text{Reduce}_{\text{Inflow}}(i); \\
S_3 : (\exists j \in \text{IN}(i) : demand(i) > 0 \land d(i) < n \land d(j) = d(i) - 1) &\Rightarrow (f(j, i) := f(j, i) + min(demand(i), r(j, i)))$;
S_4 : demand(i) > 0 \land d(i) = n \land i \neq t &\Rightarrow \text{Reduce}_{\text{Outflow}}(i)$;
\end{align*}
An illustration of the algorithm is shown in Figure 1 and Table 1.

In the legitimate state, $d(i)$ is the shortest path from $s$ to $i$ and $f$-values of the edges constitute a maximum flow. The convergence requires $O(n^2)$ moves.

<table>
<thead>
<tr>
<th>Move</th>
<th>Node</th>
<th>Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \to B$</td>
<td>$a$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>$B \to C$</td>
<td>$b$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$C \to D$</td>
<td>$t$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$D \to E$</td>
<td>$b$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$E \to F$</td>
<td>$a$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>$F \to G$</td>
<td>$b$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$G \to H$</td>
<td>$b$</td>
<td>$S_4$</td>
</tr>
<tr>
<td>$H \to I$</td>
<td>$t$</td>
<td>$S_1$</td>
</tr>
</tbody>
</table>

Table 1: Moves and corresponding nodes and guards of Figure 1

In [80], a formal definition of routing metrics and two necessary and sufficient conditions to maximize a routing metric over a tree have been presented. The paper generalizes the maximum flow tree which it calls the maximal metric tree that can stabilize in $O(n^2)$ time.

### 6 Gradient Algorithm

The self-stabilizing gradient approach proposed by Douglas [57] mimics in spirit the recently developed self-stabilized subspace methods [58, 117]. Douglas [57] extends the previous ideas in the development of self-stabilization for gradient adaptation of orthonormal matrices to develop algorithms for instantaneous prewhitened blind separation of homogeneous signal mixtures. In the problem of blind source separation (BSS), an $n$-dimensional vector sequence $x(k)$ is assumed to be produced from an $n$-dimensional source signal vector as:

\[ x(k) = Ax(k) \]  

where $A$ is an unknown nonsingular $n \times n$ mixing matrix and the elements of $s(k)$ are zero-mean, unity variance, symmetrically distributed, and statistically independent of each other. The goal of the BSS task is to process each $x(k)$ via a linear transformation $M$ such that

\[ MA = \Phi D \]  

where $\Phi$ and $D$ are $(n \times n)$ permutation and diagonal scaling matrices, respectively. The output signal vector, $y(k) = Mx(k)$, contains scaled versions of all of the elements of $s(k)$ without crosstalk. It can be assumed without loss of generality that $E[s(k)s^T(k)] = I$, such that $R_{xx} = E[x(k)x^T(k)]$, and let $P$ be a prewhitening matrix such that

\[ PR_{xx}P^T = PAAT^T = \Gamma \Gamma^T = I \]  

where $\Gamma = PA$ is an orthonormal matrix. By defining $W(k)P = M$, the optimum solution of $W(k)$ can be shown to be of the form:

\[ W_{opt} = \Phi J\Gamma^T \]  

where $J$ is an orthogonal matrix of $\pm 1$’s.

Cardoso et al. [32] define the prewhitened BSS problem as:

\[ \text{Maximize} \, \beta \sum_{i=1}^{n} E[\phi_i(y_i(k))] \]  

such that $W(k)W^T(k) = I$.  


Figure 1: An illustration of maximum flow algorithm ($d$-values are shown below each node)

where $\beta < 0$, $\phi_i(y_i) = -\log p_i(y_i)$, and $p_i(y_i)$ is the assumed density model for the source signal extracted at the $i$-th system output. This problem is equivalent to minimizing the Kullback Leibler divergence measure given by [31]:

$$D(p_s \| \hat{p}_s) = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} p_s(s) \log \left( \frac{p_s(s)}{\hat{p}_s(s)} \right) ds_1 \cdots ds_n;$$

under the constraint in (8), where $p_s(s)$ is the source vector probability density function.

The algorithms in [57] are approximate gradient procedures in the inhomogeneous space of all $n \times n$ orthonormal matrices satisfying constraint (8), a matrix space also known as the Stiefel manifold [84]. These algorithms are self-stabilizing in the sense that the projection of back to the space of orthonormal matrices is not required i.e. the rows of the adaptive demixing matrix do not need to be periodically reorthonormalized. All small perturbations of the demixing matrices away from orthonormality do not accumulate over time. The algorithms contribute to the growing body of work on gradient algorithms within constrained parameter spaces.

7 TDMA Slot Assignment

Kulkarni et al. [111] is the first to develop self-stabilizing TDMA. The paper starts from the view of a grid topology and assumes that each nodes knows its location in the grid and thus generates a TDMA schedule. But it requires the grid mapping be same for all networks and the locations be known before the TDMA algorithm starts. Hence this algorithm is not adoptable in dynamic systems. In the self-stabilizing setting, the most studied vertex coloring
problem is $L(1,0)$. Ghosh and Karaata [73] provide an
elegant algorithm for coloring planar graphs, Sur and
Srimani [135] provide the same algorithm for bipartite
graphs. Shukla et al. [132] provide algorithms for
complete odd-degree bipartite graphs and tree graphs,
while a solution for general graph is given by Gradinariu and Tixeul [82]. Recently, Gradinariu and
Johnen [81] have presented a solution to $L(1,1)$ col-
oring, using a number of colors proportional to $n^2$,
where $n$ is the number of nodes in the system. A
common drawback of these algorithms lies behind the
strong assumption of the existence of a reliable and
powerful communication system. The algorithm by
Herman [86] is tailored to networks with sensor nodes
and computes a TDMA schedule in a distributed man-
ner. Thereby an upper bound on the number of nodes
in a neighborhood is assumed. The algorithm consists
of five parts running concurrently in an endless loop.
Shared variables are propagated to the neighbors of a
node by message exchange in a globally synchronized
CSMA/CA slot. The whole process is expected to con-
verge locally in constant time and globally in sub linear
time.

8 Routing Algorithms

The notion of self-stabilization has recently been
adopted into routing algorithms. In [44], a hierarchical
routing algorithm from self-stabilizing point of view
is provided. Johnen et al. [100] report the first self-
stabilizing Border Gateway Protocol (BGP), the stan-
dard inter-domain routing protocol in the Internet. The
protocol is combined with an existing protocol to make
it resilient to policy conflicts. A self-stabilizing loop-
free routing algorithm is reported in [101] that is also
route preserving. Unlike the previous approaches, it
does not require that a bound on the network diameter
is known to the processors that perform the routing al-
gorithm. The self-stabilizing cluster routing algorithm
in [24] for MANET is based on link-cluster architec-
ture. The algorithm selects the clusterheads, and then
builds, in those nodes, routing tables regarding nodes
inside and outside the cluster. Shen et al. [131] give a
self-stabilizing routing protocol for publish-subscribe
system. Neighboring message routers periodically ex-
change their routing table state, and take corrective ac-
tions if (and only when) necessary. Datta et al. [42]
present the first self-stabilizing network algorithm in
the wormhole routing model, using the unidirectional
ring topology. The solution benefits from wormhole
routing by providing high throughput and low latency,
and front self-stabilization by ensuring automatic re-
silience to all possible transient failures.

9 Load and Resource Distribution

Gärtner et al. [66] propose a self-stabilizing algo-
Rithm for equal load distribution among the replicated
servers. Server load is measured in the number of ac-
cesses it receives within a certain period of time. The
vector $S_i^t$ is the state of node $i$ at real time $t$. A
phase $\theta_i$ of node $i$ is a fixed time interval of $t_i$. A
node performs a regular access pattern if and only if
the accesses of the node to other nodes (and thus local
state transitions) occur at the same instances in time
within every phase, i.e. $S_i^t = S_i^{t+\theta_i}$. The client side
of the distribution module (between client and server)
receives special server instruction messages which it
uses to guide the distribution process of subsequent requests by the application. The data structures of a client consist of two vectors: \( T \)-vector, \( C \)-vector. The contents of \( T \) describe the access pattern of the client within the last phase while \( C \) contains the accumulated access statistics from the current phase. The algorithm uses the access pattern \( T \) from the last phase to select a server for the next incoming request from the application. Each replicated server maintains a \( C \)-vector of current access statistics from the clients. From time to time the server receives a message from the load balancing module indicating that the load is currently high and that a certain amount of load (namely periodic accesses) should be moved to another server. The system converges to the following predicate and, at this state, loads (i.e. client accesses) get uniformly distributed among the servers:

\[
\forall i \in \text{clients} : i.C_t^t = i.C_{t-\theta} \land i.T_t^t = i.T_{t-\theta} \\
\forall j \in \text{servers} : j.C_t^t = j.C_{t-\theta}
\]

The self-stabilizing load balancing algorithm of Kam et al. [105] considers a ring of processors and ensures that a task will be scheduled on every functioning processor. On the other hand, Flatebo et al. [61] assumes that the load is completely determined by the tasks that are scheduled and do not depend on other processing that is being done. This algorithm works for an arbitrary network of processors and attempts to distribute tasks optimally over the system.

Ko et al. [109] present a decentralized, fully distributed, scalable protocol that places replicated resources in a network of arbitrary topology such that the furthest distance one must travel to find a particular copy of a resource is only slightly larger than optimal, and the distance between identical copies is large.

10 Global Optimization

For maximal matching, self-stabilizing algorithms are constructed in [89, 88] and are shown to run in linear time by Hedetniemi et al. [83]. A generalization is given in [75]. For maximum matching in trees, Ghosh et al. [68] introduce two-phase methodology: forming a rooted directed tree from an undirected tree in the first phase and then stabilization in a bottom-up manner in the second phase. These algorithms work for anonymous networks. Recently, for large ID-based networks, Goddard et al. [76, 77] have proposed a variation of maximal matching, called strong matching, a matching \( M \) with the added constraint that no two edges in \( M \) are joined by an edge.

Goddard et al. [77] also propose a minimal total dominating set and its generalization which is similar to [74]. A minimal dominating set must be maintained to optimize the number and locations of resource centers in a network. Dominating sets with specific properties have been used in several routing proposals for mobile networks. A synchronous self-stabilizing version of minimal domination protocol is given by Xu et al. [137]. A similar problem, called a maximal \( k \)-packing, for large networks, has been solved by Goddard et al. [77] in self-stabilizing fashion. This generalizes 2-packing of Karaata [106] and Gairing et al. [64].

In [52], a uniform dynamic self-stabilizing leader election algorithm is available. Self-stabilizing leader election has also been proposed in [20, 19, 7, 52, 90, 70] for different network topology. Self-stabilizing
methods for building minimum spanning trees in symmetric network graph and arbitrary network are available in [4] and [8], respectively. A minimal spanning tree must be maintained to minimize latency and bandwidth requirements of multicast or broadcast messages or to implement echo-based distributed algorithms [77]. Ghosh and Gupta [71] present a technique to transform dynamic programming into self-stabilizing distributed model that runs on trees for designing algorithms for optimization problems. For a tree with radius \( r \) this transformation stabilizes in \( O(r) \) rounds [52].

11 Distributed File System

Dolev and Kat [53] have given the first self-stabilizing distributed file system. The system constructs and maintains a spanning tree consisting of the servers that have volume replica and caches for the specific file volume. The design is based on a self-stabilizing maintenance of a distributed replica tree for each volume (the update algorithm); the tree provides a communication and consistency layer. File system updates use the tree to implement file read and write operations.

12 Shortest Path Problems

A self-stabilizing solution for the shortest path problem in a distributed network is provided by Huang and Lin [93]. For every node \( i \), the distance from source \( r \) to \( i \) is maintained in a local variable \( d(i) \) of \( i \). The weight of an edge \((i,j)\) is denoted by \( w(i,j) \) while \( N(i) \) denotes the neighbors of \( i \). The algorithm converges to the following predicate:

\[
\forall i \neq r, d(i) = \min_{j \in N(i)} (d(j) + w(i,j));
\]

A variation of this problem, called all-pairs shortest path, has been solved in self-stabilizing fashion by Chandrasekar and Srimani [33]. Self-stabilizing shortest path trees (distance-vector) can also be constructed from the maximal metric trees constructed by the self-stabilizing algorithm of Gouda et al. [80].

13 Exploration, Communication, and Token Circulation in the Network

Efficient self-stabilizing algorithms for searching over the network both in breadth-first manner [91, 134] and in depth-first manner [37, 38] resulting in a spanning tree of the network are available in the literature. However, different self-stabilizing algorithms for constructing spanning tree (not necessarily minimum spanning tree) are found in [35, 6, 65, 3]. The spanning tree algorithm in [17] works in wireless ad hoc network. Chaudhuri [34] proposes a self-stabilizing algorithm for minimum depth-search in a network.

Bein and Datta [23] design a self-stabilizing communication protocol in a sensor network, based on the directed diffusion method. A request for data from an initiator node is broadcast in the network, and the positive answers from the sensors are forwarded back to the initiator following a Shortest-Path-Tree construction rooted at the initiator. The protocol proposed by Awerbuch et al. [16] is the first self-stabilizing protocol for end to end communication. Its message complexity is comparable with the corresponding non-stabilizing solutions. Howell et al. [87] define a finite state message passing model which is particu-
larly appropriate for defining and reasoning about self-stabilizing protocols, due to the well known result that self-stabilizing protocols on unbounded-channel models must have infinitely many legitimate states. Dolev and Schiller [54, 56] present the first randomized algorithm for implementing self-stabilizing group communication services in an asynchronous system. Mizuno et al. [114] present two lock based self-stabilizing distributed mutual exclusion algorithms: one is a link-locking algorithm and the other is a node-locking algorithm. The quorum-based mutual exclusion algorithm in [115] scales well since it has constant synchronization delay and its message complexity is $O(\sqrt{n})$.

Johnen et al. [99] provide a space-efficient depth-first token circulation algorithm on a uniform rooted network. The single token circulation algorithm of Herman [85] converges in $n^2/2$ expected time. This protocol works for ring of identical machines. Different self-stabilizing token circulation algorithms have also been proposed in [120, 121, 43, 99, 123] for different network topology.

### 14 Conclusion

Self-stabilization has emerged as a promising paradigm for the design, maintenance, and analysis of fault tolerant distributed networks. As shown in [57], the self-stabilizing modifications also may prove useful for other problems in optimization and numerical linear algebra. This may also be adopted for congestion control, multi-commodity flow, and optimal routing (in future). All types of resource and load distribution may also be solved in this model.

One criticism of self-stabilization as a design goal is that it is too strong a property and thus either too difficult to achieve or can be achieved at the expense of other goals. The complexity analysis is complicated and becomes worse than the corresponding non-self-stabilizing algorithm. Under some strong assumptions, Abello et al. [1] show that any computational problem can be realized in a self-stabilizing fashion. On the other hand, Schneider [129] shows that there are some systems that are not, at all, compatible for self-stabilization. A system may experience some global state from where it cannot recover itself. This type of global state is called an **unsafe** state. It is pointed out in [129] that if any unsafe global state is a final state, then the system will not be able to stabilize.

### References


[130] Selic, B. Fault tolerance techniques for distributed systems. *(Staff, IBM)* (July 2004).


