Self-Stabilizing Leader Election in Highly Dynamic Networks

Karine Altisen\textsuperscript{1}  Stéphane Devismes\textsuperscript{2}  Anaïs Durand \textsuperscript{3}  Colette Johnen\textsuperscript{4}  Franck Petit\textsuperscript{5}

\textsuperscript{1} VERIMAG, UGA, Grenoble
\textsuperscript{2} MIS, UPJV, Amiens
\textsuperscript{3} LIMOS, Univ. Clermont, Clermont Ferrand
\textsuperscript{4} LaBRI, Univ. Bordeaux
\textsuperscript{5} LIP6, Sorbonne Université, Paris

November 9\textsuperscript{th} 2021, Fontainebleau
Challenge

Self-stabilization in Highly Dynamic Networks?

where topological changes are not:

\[
\begin{align*}
\text{transient} & \quad \text{an anomaly} \\
\text{but} & \quad \left\{ \begin{array}{l}
\text{intermittent} \\
\text{inherent}
\end{array} \right.
\end{align*}
\]

To tolerate both transient faults and high dynamics

Case Study: Leader Election
Objectives

Self-stabilizing leader election in highly dynamic message-passing systems

- Finding **conditions** under which **self-stabilizing leader election** can be achieved.

  We look for
  self-stabilizing algorithm for
  **general classes of dynamic networks**
  
  (e.g., we do not enforce the network to be in a particular topology at a given time)

- Finding the **limits** where self-stabilizing leader election becomes impossible?

- Studying **lower bounds** on the convergence time
Leader Election

\( n \) identified processes: \( \forall p \in V, id(p) \) is the unique identifier of \( p \)
Leader Election

$n$ identified processes: $\forall p \in V$, $id(p)$ is the unique identifier of $p$

Let $IDSET$ be the definition domain of identifiers ($|IDSET| > n$)

$\forall v \in IDSET$, 

- $v$ is a real ID if $\exists p \in V$, $id(p) = v$
- $v$ is a fake ID otherwise
Leader Election

$n$ identified processes: $\forall p \in V$, $id(p)$ is the unique identifier of $p$

Let $IDSET$ be the definition domain of identifiers ($|IDSET| > n$)

$\forall v \in IDSET$,
- $v$ is a real ID if $\exists p \in V$, $id(p) = v$
- $v$ is a fake ID otherwise

Every process $p$ computes the identifier of the leader in $lid(p)$
Initially, $lid(p)$ may contain a fake ID

**GOAL:** converge to a configuration from which all $lid$ variables constantly designates the same real ID
Computation Model

- **Synchronous Rounds:**

- **Dynamics** modeled with a **Dynamic Graph (DG)**

[Xuan et. al., 03], [Casteigts et. al., 13]
Can d transmit information to a?
Can \(d\) transmit information to \(a\)?
Can \(d\) transmit information to \(a\) ?

1, \((d, c)\); 2, \((c, b)\)
Can \(d\) transmit information to \(a\) ?

\[1, (d, c); 2, (c, b)\]
Journey

Can $d$ transmit information to $a$?

$G_1 \xrightarrow{\sim \sim} G_2 \xrightarrow{\sim \sim} G_3 \xrightarrow{\sim \sim} G_4 \xrightarrow{\sim \sim} \ldots$

1, $(d, c)$; 2, $(c, b)$; 4, $(b, a) = \text{Journey from } d \text{ to } a \text{ of length 4}.$
Sources and Sinks

**Source:** can infinitely often reach any other through a journey

**Sink:** can infinitely often be reached by any other through a journey
Sources and Sinks

Source: can infinitely often reach any other through a journey

Quasi-Timely Source: can infinitely often reach any other through a journey of length $\leq \Delta$

Timely Source: can always reach any other through a journey of length $\leq \Delta$
Sources and Sinks

**Source:** can infinitely often reach any other through a journey

**Quasi-Timely Source:** can infinitely often reach any other through a journey of length $\leq \Delta$

**Timely Source:** can always reach any other through a journey of length $\leq \Delta$
Sources and Sinks

Source: can infinitely often reach any other through a journey

Quasi-Timely Source: can infinitely often reach any other through a journey of length $\leq \Delta$

Timely Source: can always reach any other through a journey of length $\leq \Delta$

Sink: can infinitely often be reached by any other through a journey

Quasi-Timely Sink: can infinitely often be reached by any other through a journey of length $\leq \Delta$

Timely Sink: can always be reached by any other through a journey of length $\leq \Delta$
Classes where All processes are Sources (and so Sinks)

\[ \mathcal{I}_{*,*} : \text{All processes are sources} \]
\[ \mathcal{I}_{*,*}^Q(\Delta) : \text{All processes are quasi-timely sources} \]
\[ \mathcal{I}_{*,*}^B(\Delta) : \text{All processes are timely sources} \]
Generalization: Classes with at least One Source or One Sink

$\mathcal{I}_{1,*}$: At least one Source

$\mathcal{I}_{1,*}^{Q}(\Delta)$: At least one Quasi-Timely Source

$\mathcal{I}_{1,*}^{B}(\Delta)$: At least one Timely Source

$\mathcal{I}_{*,1}$: At least one Sink

$\mathcal{I}_{*,1}^{Q}(\Delta)$: At least one Quasi-Timely Sink

$\mathcal{I}_{*,1}^{B}(\Delta)$: At least one Timely Sink
Hierarchy

\[ \mathcal{J}_1, \ast \rightarrow \mathcal{J}_1, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ \mathcal{J}_1, \ast \rightarrow \mathcal{J}_Q, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ \mathcal{J}_Q, \ast \rightarrow \mathcal{J}_1, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ \mathcal{J}_1, \ast \rightarrow \mathcal{J}_Q, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ \mathcal{J}_Q, \ast \rightarrow \mathcal{J}_1, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ \mathcal{J}_1, \ast \rightarrow \mathcal{J}_Q, \ast \rightarrow \mathcal{J}_1, \ast \]

\[ A \rightarrow B \text{ means that } A \subset B \]
Main Results

Self-stabilization
Convergence Time Boundable only in \( J^{B}_*, * \)

Pseudo-stabilization

Convergence Time Unboundable

Self-stabilization

Pseudo-stabilization

Self-stabilization
Convergence Time Boundable only in \( J^{B}_*, * \)
Self- vs. Pseudo-stabilization: "cannot" vs. "does not"

- **Self-stabilization:**

  - arbitrary initial config.
  - legitimate config.

- **Pseudo-stabilization:**

  - arbitrary initial config.
Classes where All Processes are Sources

$\mathcal{T}C^B(\Delta) \subseteq \mathcal{T}C^Q(\Delta) \subseteq \mathcal{T}C^R$
Class $\mathcal{TC}^B(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Bounded Temporal Diameter):

- $\Delta$ known
- Stabilization Time: at most $3\Delta$ rounds
- Memory Requirement: $O(\log n + \log \Delta)$ bits per node

Class $\mathcal{TC}^Q(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Quasi Bounded Temporal Diameter):

- $\Delta$ and $n$ known
- Memory Requirement: $O(n(\log n + \log \Delta))$ bits per node

Class $\mathcal{TC}^R$ (Recurrent Temporal Connectivity):

- $n$ known
- Memory Requirement: infinite

For $\mathcal{TC}^Q(\Delta)$ and $\mathcal{TC}^R$, the convergence time cannot be bounded and knowledge of $n$ is mandatory!
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$. 
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $\mathcal{A}$ that “knows” the number of processes $n$ in the system.
Let $p_1$, $p_2$, $p_3$, $p_4$, $p_5$, and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the number of processes $n$ in the system.

If $A$ runs on $p_1$, $p_2$, $p_3$, $p_4$, $p_5$, and $p_6$:
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that "knows" the number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:

If $A$ runs on $p_1, p_2, p_3,$ and $p_4$:

$$\text{size-ambiguity (1/3)}$$
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that "knows" the number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:

It is not size-ambiguous!

If $A$ runs on $p_1, p_2, p_3,$ and $p_4$:
Let $p_1, p_2, p_3, p_4, p_5, \text{ and } p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $\mathcal{A}$ that “knows” the parity of number of processes $n$ in the system.
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $\mathcal{A}$ that “knows” the parity of number of processes $n$ in the system.

If $\mathcal{A}$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the parity of number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
<th>$p_5$</th>
<th>$p_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If $A$ runs on $p_1, p_2, p_3,$ and $p_4$:

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_3$</th>
<th>$p_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

It is size-ambiguous!
Let $p_1, p_2, p_3, p_4, p_5$, and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the parity of number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5$, and $p_6$:

It is size-ambiguous!

If $A$ runs on $p_1, p_2, p_3$, and $p_4$:
Let \( p_1, p_2, p_3, p_4, p_5, \) and \( p_6 \) be a set of processes. The identifier of \( p_i \) is \( i \).

Assume an algorithm \( A \) that “knows” the bound \( K = 9 \) of number of processes \( n \) in the system.
Let $p_1, p_2, p_3, p_4, p_5$, and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the bound $K = 9$ of number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5$, and $p_6$: 

1. $p_1^9$
2. $p_2^9$
3. $p_3^9$
4. $p_4^9$
5. $p_5^9$
6. $p_6^9$
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the bound $K = 9$ of number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:

If $A$ runs on $p_1, p_2, p_3,$ and $p_4$:
Let $p_1, p_2, p_3, p_4, p_5,$ and $p_6$ be a set of processes. The identifier of $p_i$ is $i$.

Assume an algorithm $A$ that “knows” the bound $K = 9$ of number of processes $n$ in the system.

If $A$ runs on $p_1, p_2, p_3, p_4, p_5,$ and $p_6$:

![Diagram](Diagram1)

If $A$ runs on $p_1, p_2, p_3,$ and $p_4$:

![Diagram](Diagram2)

It is size-ambiguous!


**Size-ambiguity (definition)**

Let $V$ be a set of processes and $k \in \mathbb{N}$.

$\mathcal{A}$ is $(k, V)$-ambiguous if $0 < k < |V|$ and for every $U \subset V$ such that $|U| = k$, $\mathcal{A}$ can be run on $U$ and for every $p \in U$, $p$ has the same set of states whether $\mathcal{A}$ runs on $U$ or $V$.

$\mathcal{A}$ is size-ambiguous if there exists $V$ and $k$ such that $\mathcal{A}$ is $(k, V)$-ambiguous.
Let $V$ be a set of processes and $k \in \mathbb{N}$.

$\mathcal{A}$ is $(k, V)$-ambiguous if $0 < k < |V|$ and for every $U \subseteq V$ such that $|U| = k$, $\mathcal{A}$ can be run on $U$ and for every $p \in U$, $p$ has the same set of states whether $\mathcal{A}$ runs on $U$ or $V$.

$\mathcal{A}$ is size-ambiguous if there exists $V$ and $k$ such that $\mathcal{A}$ is $(k, V)$-ambiguous.

$\mathcal{A}$ is size-ambiguous $\approx$ “$\mathcal{A}$ has a partial knowledge of $n$”
Let $\mathcal{A}$ be any self-stabilizing leader election algorithm for $\mathcal{TC}^Q(\Delta)$ ($\Delta \geq 2$), $V$ be a set of processes, $\mathcal{L}$ be a set of legitimate configurations of $\mathcal{A}$ for $V$, and $k \in \mathbb{N}$.

$\mathcal{L}$ is closed in $\mathcal{TC}^Q(\Delta)$: if $\gamma'$ reachable from $\gamma \in \mathcal{L}$ by $\mathcal{A}$ in $\mathcal{TC}^Q(\Delta)$, $\gamma' \in \mathcal{L}$ too.

- If $\mathcal{A}$ is $(k, V)$-ambiguous, then $\mathcal{L}$ is not closed in $\mathcal{TC}^Q(\Delta)$. (also holds for $\mathcal{TC}^B(\Delta)$)
- $\exists$ a set of legitimate configurations of $\mathcal{A}$ for $V$ which is closed in $\mathcal{TC}^Q(\Delta)$.
Let $\mathcal{A}$ be any self-stabilizing leader election algorithm for $\mathcal{T}C^Q(\Delta)$ ($\Delta \geq 2$), $V$ be a set of processes, $\mathcal{L}$ be a set of legitimate configurations of $\mathcal{A}$ for $V$, and $k \in \mathbb{N}$.

$\mathcal{L}$ is closed in $\mathcal{T}C^Q(\Delta)$: if $\gamma'$ reachable from $\gamma \in \mathcal{L}$ by $\mathcal{A}$ in $\mathcal{T}C^Q(\Delta)$, $\gamma' \in \mathcal{L}$ too.

- If $\mathcal{A}$ is $(k, V)$-ambiguous, then $\mathcal{L}$ is not closed in $\mathcal{T}C^Q(\Delta)$. (also holds for $\mathcal{T}C^B(\Delta)$)
- $\exists$ a set of legitimate configurations of $\mathcal{A}$ for $V$ which is closed in $\mathcal{T}C^Q(\Delta)$.

**Theorem 1**

*No self-stabilizing leader election algorithm for $\mathcal{T}C^Q(\Delta)$, with $\Delta \geq 2$, can be size-ambiguous.*

**Corollary 2**

*No self-stabilizing leader election algorithm for $\mathcal{T}C^R$ can be size-ambiguous.*
Other Classes: Focus on $\mathcal{J}_{1,*}^B(\Delta)$

At least one a priori unknown process (a timely source) can always reach any other through a journey of length $\leq \Delta$
Impossibility of Self-stabilizing Leader Election in $\mathcal{J}_{1,*}^B(\Delta)$

**Preliminary Result**

In situation $\mathcal{A}$, one process eventually changes its leader output.

![Diagram](image)

**Proof:**

- All processes except $p_i$ (resp. $p_x$) are connected to any other at any time $\Rightarrow \in \mathcal{J}_{1,*}^B(\Delta)$

- $\forall j \notin \{i, x\}$, the executions of $p_j$ in $\mathcal{A}$ and $\mathcal{B}$ are **indistinguishable**

  $\Rightarrow$ if $p_j$ elects $p_i$ in $\mathcal{A}$, then $p_j$ elects $p_i$ in $\mathcal{B} \Rightarrow \times$

  $\Rightarrow p_j$ eventually changes its leader
Assume a self-stabilizing algorithm exists

1. Starting from any legitimate configuration, \( \text{lid} \) variables should be constant
2. Now, from any legitimate configuration, situation \( \mathcal{A} \) is possible
   
   Preliminary result \( \Rightarrow \) one process eventually changes its leader

Contradiction
Assume a self-stabilizing algorithm exists

Starting from any legitimate configuration, \( \text{lid} \) variables should be constant.

Now, from any legitimate configuration, situation \( \mathcal{A} \) is possible.

Preliminary result \( \Rightarrow \) one process eventually changes its leader

Contradiction
Goal: Electing a "stable" process

**Stable Process:** eventually, all other processes receive (maybe indirectly) information about it at least every $\Delta$ rounds
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^B(\Delta)$

Goal: Electing a "stable" process

**Stable Process:** eventually, all other processes receive (maybe indirectly) information about it at least every $\Delta$ rounds

At each round, each process initiates a **flooding** (relayed $\Delta$ times)

→ ∃ **stable processes**: each source is stable!
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^B(\Delta)$

Goal: Electing a "stable" process

**Stable Process:** eventually, all other processes receive (maybe indirectly) information about it at least every $\Delta$ rounds

At each round, each process initiates a *flooding* (relayed $\Delta$ times)

$\implies$ *stable processes*: each source is stable!

How to evaluate stability?
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^B(\Delta)$

Goal: Electing a "stable" process

**Stable Process:** eventually, all other processes receive (maybe indirectly) information about it at least every $\Delta$ rounds

At each round, each process initiates a **flooding** (relayed $\Delta$ times)

$\rightarrow \exists$ **stable processes**: each source is stable!

How to evaluate stability? **suspicion counter**

A process increments its suspicion counter each time it is accused to be NOT stable by some process

**After the 1st round**, suspicion counters are **monotonically non-decreasing**

(a counter may be reset during the first round due to initial inconsistency)
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,\ast}^B(\Delta)$

**Goal:** Electing a "stable" process

**Stable Process:** eventually, all other processes receive (maybe indirectly) information about it at least every $\Delta$ rounds

At each round, each process initiates a *flooding* (relayed $\Delta$ times)

$\rightarrow \exists$ stable processes: each source is stable!

How to evaluate stability? **suspicion counter**

A process increments its suspicion counter each time it is accused to be NOT stable by some process

**After the 1st round**, suspicion counters are *monotonically non-decreasing*

(a counter may be reset during the first round due to initial inconsistency)

**Elected Leader:** a process with the minimum suspicion counter value

(we use identifiers to break ties)
Each process $p$ maintains two maps:

- **$LStable(p)$**: Map of *locally stable* processes at $p$  
  $\Rightarrow$ $p$ itself and processes from which $p$ (directly) receives information at most $\Delta$ rounds ago.

- **$GStable(p)$**: Map of *globally stable* processes  
  $=$ locally stable at any process ($p$ included)  
  $\Rightarrow$ must eventually contain at least every stable process.
Locally and Globally Stable Processes

Each process $p$ maintains two maps:

- **$LStable(p)$**: Map of *locally stable* processes at $p$
  $\Rightarrow$ $p$ itself and processes from which $p$ (directly) receives information at most $\Delta$ rounds ago.

- **$GStable(p)$**: Map of *globally stable* processes
  $=\text{locally stable at any process (}p\text{ included)}$
  $\Rightarrow$ must eventually contain at least every stable process

$p$ always considers *itself* locally & globally stable
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{F}_{1,*}^{B}(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $< id, susp, ttl >$

- $id$: identifier
- $susp$: the suspicion value of $id$
- $ttl$: time to live
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^B(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $<id, susp, ttl>$

- $id$: identifier
- $susp$: the suspicion value of $id$
- $ttl$: time to live

$LStable(p)$ and $GStable(p)$ are appended/updated with received information

- update in $LStable(p)$: information with the highest $ttl$ is considered as the freshest one
- update in $GStable(p)$: received information is considered as fresher and inserted with $ttl = \Delta$
Pseudo-stabilizing Leader Election Algorithm for $J_{1,*}^B(\Delta)$

Locally and Globally Stable Processes

Values inside $LStable(p)$ and $GStable(p)$: triplet $< id, susp, ttl >$

- $id$: identifier
- $susp$: the suspicion value of $id$
- $ttl$: time to live

$LStable(p)$ and $GStable(p)$ are appended/updated with received information

- update in $LStable(p)$: information with the highest $ttl$ is considered as the freshest one
- update in $GStable(p)$: received information is considered as fresher and inserted with $ttl = \Delta$

A triplet is removed from a map when its $ttl$ reaches 0
At Every Round

1. \( p \) initiates a flooding of the triplet 
   \(< id(p), LSP = LStable(p), ttl = \Delta >\)
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^B_{1,*}(\Delta)$

At Every Round

1. $p$ initiates a flooding of the triplet 
   \[< id(p), LSP = LStable(p), ttl = \Delta >\]

2. $p$ updates $LStable(p)$ and $GStable(p)$ according to received triplets
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^{B}(\Delta)$

At Every Round

1. $p$ initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$

2. $p$ updates $LStable(p)$ and $GStable(p)$ according to received triplets

3. $ttl$ variables (except those associated to $p$) are decremented and expired triplets are deleted
At Every Round

1. $p$ initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$

2. $p$ updates $LStable(p)$ and $GStable(p)$ according to received triplets

3. $ttl$ variables (except those associated to $p$) are decremented and expired triplets are deleted

4. For each received $LSP$ map, if $id(p)$ is absent from $LSP$, $p$ increments its suspicion counter
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}^{B}_{1,*}(\Delta)$

At Every Round

1. $p$ initiates a flooding of the triplet $< id(p), LSP = LStable(p), ttl = \Delta >$

2. $p$ updates $LStable(p)$ and $GStable(p)$ according to received triplets

3. $ttl$ variables (except those associated to $p$) are decremented and expired triplets are deleted

4. For each received $LSP$ map, if $id(p)$ is absent from $LSP$, $p$ increments its suspicion counter

5. $p$ elects $q \in GStable(p)$ with lowest suspicion counter
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^{B}(\Delta)$

Pseudo-stabilization

1. "Time To Live" allow to delete fake IDs.
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{I}_{1,*}^B(\Delta)$

Pseudo-stabilization

1. "Time To Live" allow to delete fake IDs.

2. Let $s$ be a source.
   
   $p \neq s$ receives $< id(s), LSP, ttl >$ at least every $\Delta$ rounds
   
   $\rightarrow$ eventually $id(s) \in LStable(p)$ forever, $\forall p \in V$
   
   $\Rightarrow$ the suspicion counter of $s$ is eventually forever constant
   
   $\rightarrow id(s) \in LSP$
   
   $\Rightarrow$ eventually $id(s) \in GStable(p)$ forever, $\forall p \in V$
**Pseudo-stabilizing Leader Election Algorithm for $J_{1,1}^B(\Delta)$**

**Pseudo-stabilization**

1. **"Time To Live"** allow to delete fake IDs.

2. Let $s$ be a source.
   
   $p \neq s$ receives $< id(s), LSP, ttl >$ at least every $\Delta$ rounds
   
   → eventually $id(s) \in LStable(p)$ forever, $\forall p \in V$
   
   ⇒ the suspicion counter of $s$ is eventually forever constant
   
   → $id(s) \in LSP$
   
   ⇒ eventually $id(s) \in GStable(p)$ forever, $\forall p \in V$

3. Let $x$ be a process whose suspicion counter is eventually constant
   
   Eventually $id(x) \in LStable(s)$ forever, for every source $s$
   
   → $id(x) \in GStable(p), \forall p \in V$
Pseudo-stabilizing Leader Election Algorithm for $J^B_{1,*}(\Delta)$

Pseudo-stabilization

4. $x$ infinitely often absent of $GStable(p)$
   $\implies$ infinitely often, during $\Delta$ consecutive rounds,
   $p$ only receives $<-, LSP, ->$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources
   $\rightarrow x$ also receives these latter
   $\rightarrow x$ increments its counter infinitely often
Pseudo-stabilizing Leader Election Algorithm for $J_{1,*}^B(\Delta)$

**Pseudo-stabilization**

4. $x$ infinitely often absent of $GStable(p)$
   $\implies$ infinitely often, during $\Delta$ consecutive rounds,
   $p$ only receives $<-, LSP, ->$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources
   $\rightarrow x$ also receives these latter
   $\rightarrow x$ increments its counter infinitely often

5. Eventually:
   Processes with **eventually const.** susp. counter (at least 1) $\in GStable(p)$ forever
   Suspicion counter of other processes $> constant$ suspicion counters
Pseudo-stabilizing Leader Election Algorithm for $\mathcal{J}_{1,*}^B(\Delta)$

Pseudo-stabilization

4. $x$ infinitely often absent of $GStable(p)$

$\implies$ infinitely often, during $\Delta$ consecutive rounds,

$p$ only receives $< -, LSP, -$ with $id(x) \notin LSP$

Some of those triplets were initiated by sources

$\implies x$ also receives these latter

$\implies x$ increments its counter infinitely often

5. Eventually:

Processes with eventually const. susp. counter (at least 1) $\in GStable(p)$ forever

Suspicion counter of other processes $> constant$ suspicion counters

6. Eventually, $\ell \in GStable(p)$ with lowest suspicion counter is the same at every $p$

$\implies$ every process elects $\ell$, a stable process.
Conclusion
When stabilization is possible, \textit{convergence time} is, most of the time, unboundable ...

Notable exception: $\mathcal{J}_{*,*}^{(\Delta)}$
When stabilization is possible, *convergence time* is, most of the time, unboundable ...

Notable exception: $J_{*,*}^{B}(\Delta)$

However, to mitigate this issue, we have *speculation*
Speculation [Kotla et al., ACM Trans. Comput. Syst., 2009]:

- the system satisfies its requirements for all executions,
- but also exhibits significantly better performances in a subset of more probable executions.

Idea: worst possible scenarios are often rare in practice.

When the stabilization time cannot be bounded in a class, we exhibit an important subclass where it can be bounded.
Speculation [Kotla et al., ACM Trans. Comput. Syst., 2009]:
- the system satisfies its requirements for all executions,
- but also exhibits significantly better performances in a subset of more probable executions.

Idea: worst possible scenarios are often rare in practice.

When the stabilization time cannot be bounded in a class, we exhibit an important subclass where it can be bounded.

In all cases where stabilization is possible but the convergence time is unboundable, our algorithms are speculative: when deployed in the subclass $J^{B}_\star,\star(\Delta)$, the convergence time is in $O(\Delta)$ rounds.
Important open questions:

- Can we solve pseudo-stabilizing leader election in $J_{1,*}^B$ with a bounded memory?

- Can we solve self-stabilizing leader election in $J_{*,*}$ with a bounded memory?
Thank You for Your Attention

Publications:


- Karine Altisen, Stéphane Devismes, and Anaïs Durand, Colette Johnen, and Franck Petit. On Implementing Stabilizing Leader Election with Weak Assumptions on Network Dynamics. *PODC’21*. 