Bounding the impact of unbounded attacks

Sébastien Tixeuil (UPMC)
joint work with
Toshimitsu Masuzawa (大阪大学)

ALADDIN meeting, 30 May 2008
supported in part by SAKURA and ALADDIN
Bounding the impact of unbounded attacks

Sébastien Tixeuil (UPMC)
joint work with
Toshimitsu Masuzawa (大阪大学)

ALADDIN meeting, 30 May 2008
supported in part by SAKURA and ALADDIN
Outline

1. Stabilization vs. permanent faults
   - Strict stabilization
   - Strong stabilization

2. Strongly-stabilizing protocols
   - System model
   - Tree orientation
   - Tree construction

3. Conclusions
Self-stabilization

- Self-stabilizing protocol
  - starts from any configuration, but
  - always converge to its intended behavior

⇒ It can tolerate any transient faults or attacks

arbitrary configuration
processes at arbitrary states
Stabilization vs. permanent faults

- Self-stabilizing protocol
  - convergence to its intended behavior is guaranteed only when no fault occurs during convergence

- Large-scale distributed system
  - some faults may always exist

- Required additional property
  - Correct processes converge to their intended behaviors despite permanent faults
Stabilization vs. permanent faults

- Self-stabilizing protocol
  - Convergence to its intended behavior is guaranteed only when no fault occurs during convergence

- Large-scale distributed system
  - Some faults may always exist

- Required additional property
  - Correct processes converge to their intended behaviors despite permanent faults
Stabilization vs. permanent faults

- Self-stabilization
  - Recovery from an arbitrary configuration
    - Good adaptability to dynamical changes

- (Permanent) **Malicious process**
  - Arbitrary behaviour
    - Continuous changes of the process state
Stabilization vs. permanent faults

- **Self-stabilization**
  - Recovery from an arbitrary configuration
    - Good adaptability to dynamical changes

- **(Permanent) Malicious process**
  - Arbitrary behaviour
    - Continuous changes of the process state
Stabilization vs. permanent faults

- **Self-stabilization**
  - Recovery from an arbitrary configuration
    - Good adaptability to dynamical changes

- (Permanent) **Malicious process**
  - Arbitrary behaviour
    - Continuous changes of the process state

Stabilization is difficult to attain.
Strict stabilization

- **Strict stabilization** (Nesterenko, Arora 2002)
  - Restrict the range of contaminated processes
    - **Containment radius (C-radius)**
      - Largest distance to contaminated processes
    - No influence outside the C-radius
Strict stabilization

- **Strict stabilization** (Nesterenko, Arora 2002)
  - Restrict the range of contaminated processes
    - **Containment radius (C-radius)**
      - Largest distance to contaminated processes
    - No influence outside the C-radius
Strict stabilization

- **Strict stabilization** (Nesterenko, Arora 2002)
  - Restrict the range of contaminated processes
    - **Containment radius (C-radius)**
      Largest distance to contaminated processes
  - No influence outside the C-radius

- **Inside C-radius**, a process may permanently violate the problem specification
- **Outside C-radius**, a process eventually reaches a config. after which it keeps satisfying the problem specification
Previous results

- Nesterenko, Arora (2002)
  - vertex coloring
    - $C$-radius=1
  - dining philosophers
    - $C$-radius=2

- Sakurai, Ooshita, Masuzawa (2004)
  - link coloring on trees
    - $C$-radius=2

  - link coloring on arbitrary networks
    - $C$-radius=1
Exercise: node coloring

- Assume at least degree+1 colors

- Algorithm 1
  - Pick minimal available color among neighbors

- Algorithm 2
  - Pick whatever available color among neighbors

- What about strict stabilization?
The problem

- r-restrictive tasks
  - A task is r-restrictive if its specification forbids combinations of processors states that are distance r from each other
- Theorem [Nesterenko, Arora, 2002]
  - The C-radius of a r-restrictive task is at least r

- No hope for “global” problems?
  - tree orientation
  - tree construction
Strong stabilization

- **Strong stabilization**
  - **Containment radius (C-radius)**
  - Allow processes outside the C-radius to be disturbed but only a bounded number of times
  - **Containment times (C-times)**
    The largest number of times of being disturbed at a process
Strong stabilization

- **Strong stabilization**
  - Containment radius (*C*-radius)
  - Allow processes outside the *C*-radius to be **disturbed** but only a **bounded** #times
  - Containment times (*C*-times)
    - The largest #times of being disturbed at a process
Strong stabilization

- Strong stabilization
  - Containment radius (C-radius)
  - Allow processes outside the C-radius to be disturbed but only a bounded times
  - Containment times (C-times)
    - The largest times of being disturbed at a process

- No bound on the time when the last disturbance occurs!!
Strong stabilization

- Strong stabilization
  - Containment radius (C-radius)
  - Allow processes outside the C-radius to be disturbed but only a bounded \#times
  - Containment times (C-times)
    - The largest \#times of being disturbed at a process

✓ outside C-radius, a process may never reach a config. after which it keeps satisfying the problem specification but eventually keeps satisfying the problem specification
Strong stabilization vs. Strict stabilization

Malicious

Inside the C-radius

Strict stab.

Outside the C-radius

Strong stab.

Outside the C-radius

problem specification is satisfied
Strong stabilization vs. Strict stabilization

Malicious

Inside the C-radius

Strict stab.

Outside the C-radius

Strong stab.

problem specification is satisfied

Each process is disturbed at most $t$ times
Strong stabilization

- **Problem**
  - defined by predicate on the output variables \((O\text{-}var)\)
  - \((t, c, f )\)-time contained configuration
    - \((t : C\text{-}times, c : C\text{-}radius, f : \#Byzantines)\)
    - In any execution starting from it, each process outside the \(C\text{-}radius\)
      - changes its \(O\text{-}var\) at most \(t\) times, and
      - eventually keep satisfying the problem requirement
  - \((t, c, f )\)-strongly stabilizing protocol
    - eventually reaches a \((t, c, f )\)-time contained config.
Strong stabilization

- $(t, c, f)$-strongly stabilizing protocol

$(t : C\times, c : C\text{-radius}, f : \#\text{Byzantines})$
Strong stabilization

- \((t, c, f)-\text{strongly stabilizing}\
  \text{protocol}\)

\((t : \text{C-times}, c : \text{C-radius}, f : \#\text{Byzantines})\)

\(\text{Malicious}\)

\(\text{Inside the C-radius}\)

\(\text{Outside the C-radius}\)

\((t, c, f)-\text{time contained}\)
Strong stabilization

- $(t, c, f)$-strongly stabilizing protocol

$(t: C$-times, $c: C$-radius, $f: \#Byzantines)$

Malicious

Each process changes its state at most $t$ times
Strong stabilization

- \((t, c, f)\)-strongly stabilizing protocol

\((t : C\text{-}times, c : C\text{-}radius, f : \#\text{Byzantines})\)

Malicious

\((t, c, f)\)-time contained

Inside the C-radius

problem specification is satisfied

Outside the C-radius

Each process changes its state at most \( t \) times
Outline

1. Stabilization vs. permanents faults
   - Strict stabilization
   - Strong stabilization

2. Strongly-stabilizing protocols
   - System model
   - Tree orientation
   - Tree construction

3. Conclusions
System model

- **tree networks**
  - non-rooted tree
  - no unique ID
  - each process can distinguish its neighbors from each other

- **link-register model**

- **distributed daemon**
  - Any subset of processes can be simultaneously activated
System model

- **tree networks**
  - non-rooted tree
  - no unique ID
  - each process can distinguish its neighbors from each other

- **link-register model**

- **distributed daemon**
  - Any subset of processes can be simultaneously activated
Tree orientation

- From a non-rooted tree to a rooted tree with a root link
  - Each process selects a neighbor as its parent
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node** (next to Byzantine)

**Fault-free case**

**Faulty case**
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node** (next to Byzantine)

**Fault-free case**

**Faulty case**
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node (next to Byzantine)**

**Fault-free case**

**Faulty case**
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node** (next to Byzantine)

**Fault-free case**

**Faulty case**
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node** (next to Byzantine)

**Fault-free case**

**Faulty case**
Tree orientation

- From a **non-rooted** tree to a **rooted** tree
- Each process selects a neighbor as its parent so that each **non-faulty fragment** forms
  - a rooted tree with a **root link**, or
  - a rooted tree with a **root node** (next to Byzantine)

**Fault-free case**

**Faulty case**
Our results for tree orientation

- **Strong stabilization for tree orientation**
  - Case of two Byzantines
    - Impossibility of
      - $C$-radius $= o(n)$ and $C$-times $= \text{bounded}$
  - Case of one Byzantine
    - Possibility of
      - $C$-radius $= 0$ and $C$-times $= 1$
        - optimal

Our results for tree orientation

- **Strong stabilization for tree orientation**
  - Case of **two** Byzantines
    - **Impossibility** of
      - $C$-radius = $o(n)$ and $C$-times = bounded
  - Case of **one** Byzantine
    - **Possibility** of
      - $C$-radius = 0 and $C$-times = 1
      - optimal
      - optimal

From characterization of **strict** stabilization

- **Impossibility** of
  - $C$-radius = $o(n)$ and $C$-times = 0
Impossibility with two Byzantines

- No strongly-stabilizing protocol, resilient to two Byzantines, with
  - $C$-radius of $o(n)$, and
  - bounded $C$-times ($n : \#\text{processes}$)

✓ The impossibility holds even for the central daemon
  - No two processes can be simultaneously activated
Impossibility with two Byzantines

\( S \): a line of \( n \) processes with two Byzantine ends

\( S' \): a line of \( 3n \) processes with no Byzantine

1. Construct a config. of \( S' \) from a legitimate config. of \( S \).
Impossibility with two Byzantines

$S$: a line of $n$ processes with two Byzantine ends

$S'$: a line of $3n$ processes with no Byzantine

1. Construct a config. of $S'$ from a legitimate config. of $S$. 

$C$-radius $o(n)$
Impossibility with two Byzantines

$S$: a line of $n$ processes with two Byzantine ends
$S'$: a line of $3n$ processes with no Byzantine

1. Construct a config. of $S'$ from a legitimate config. of $S$. 

$C$-radius $o(n)$
Impossibility with two Byzantines

$S$: a line of $n$ processes with two Byzantine ends
$S'$: a line of $3n$ processes with no Byzantine

1. Construct a config. of $S'$ from a legitimate config. of $S$.
2. One of $v1$, $v2$ and $v3$ changes its parent.
Proof argument

1. Construct a config. of $S'$ from a legitimate config. of $S$.
2. One of $v1$, $v2$ and $v3$ changes its parent.
Proof argument

1. Construct a config. of $S'$ from a legitimate config. of $S$.
2. One of $v_1$, $v_2$ and $v_3$ changes its parent.
3. Construct an execution of $S$ where $v$ changes its parent infinitely often.
Proof argument

1. Construct a config. of $S'$ from a legitimate config. of $S$.
2. One of $v_1$, $v_2$ and $v_3$ changes its parent.
3. Construct an execution of $S$ where $v$ changes its parent infinitely often.
Proof argument

1. Construct a config. of $S'$ from a legitimate config. of $S$.
2. One of $v_1$, $v_2$ and $v_3$ changes its parent.
3. Construct an execution of $S$ where $v$ changes its parent infinitely often.
Strongly-stabilizing tree orientation?

- Most common approach for tree orientation
  - A link incident to a center process becomes the root link.
Strongly-stabilizing tree orientation?

- Most common approach for tree orientation
  - A link incident to a center process becomes the root link.
Strongly-stabilizing tree orientation?

- Most common approach for tree orientation
  - A link incident to a center process becomes the root link.
Strongly-stabilizing tree orientation?

- Most common approach for tree orientation
  - A link incident to a center process becomes the root link.

This approach cannot contain a single Byzantine.
**Strongly-stabilizing tree orientation?**

- **Most common approach for tree orientation**
  - A link incident to a center process becomes the root link.

  This approach **cannot contain a single Byzantine**.

---

Byzantine process can **pull and push the center**.
Strongly-stabilizing tree orientation

- Restrict to **one-sided effect**
  - The Byzantine *can pull* but *cannot push* the root
- **Our protocol** (action of process v)
  1. If there is a neighbor u s.t. 
     \[ \text{level}(u) > \text{level}(v), \]
     then \( \text{parent}(v) = u; \text{level}(v) = \text{level}(u) \)

     ![Diagram 1](image1)

  2. If there is a neighbor u s.t.
     
     \[ \text{level}(u) = \text{level}(v), \text{parent}(u) = /v \text{ and parent}(v) = u, \]
     then \( \text{parent}(v) = u; \text{level}(v) = \text{level}(u)+1 \)

     ![Diagram 2](image2)
Legitimate configurations

- The protocol reaches a config. where each fragment of correct processes satisfies C1 or C2

**C1: (strictly legitimate)**
- A rooted tree with the **root node** v adjacent to the Byzantine process z is formed,
- `parent(v) = z`, and
- `level(w) ≥ level(x)` holds for any parent w and its child x

**C2: (legitimate)**
- A rooted tree with a **root link** is formed, and
- all processes have the **same level**
Legitimate condition C1

Strictly legitimate

- A rooted tree with the root node $v$ adjacent to the Byzantine process $z$ is formed,
- parent($v$) = $z$, and
- $\text{level}(w) > \text{level}(x)$ holds for any parent $w$ and its child $x$. 

![Diagram of a rooted tree with node $v$ adjacent to Byzantine process $z$ and level values for various nodes.]
Legitimate condition C1

Strictly legitimate

- A rooted tree with the root node $v$ adjacent to the Byzantine process $z$ is formed,
- $\text{parent}(v) = z$, and
- $\text{level}(w) > \text{level}(x)$ holds for any parent $w$ and its child $x$. 

Decrease: no influence
Increase: influence only on levels
Legitimate condition C1

Strictly legitimate

- A rooted tree with the root node $v$ adjacent to the Byzantine process $z$ is formed,
- $\text{parent}(v) = z$, and
- $\text{level}(w) > \text{level}(x)$ holds for any parent $w$ and its child $x$

Once a fragment satisfies C1, C1 holds forever

Decrease: no influence
Increase: influence only on levels
Legitimate condition C2

- A rooted tree with the **root link** is formed, and
- all processes have the **same level**
Legitimate condition C2

- A rooted tree with the root link is formed, and
- all processes have the same level
Legitimate condition C2

- A rooted tree with the root link is formed, and
- all processes have the same level
Legitimate condition C2

- A rooted tree with the root link is formed, and
- all processes have the same level

Increase causes transition to a strictly legitimate config.
Legitimate condition C2

- A rooted tree with the *root link* is formed, and
- all processes have the *same level*

Increase causes transition to a *strictly legitimate* config.

From C2 to C1, each process *changes its parent at most once.*

\[ C\text{-times} = 1 \]
An execution example

initial config.
An execution example

initial config.
An execution example

initial config.
An execution example

initial config.

strictly legitimate
An execution example

initial config.

strictly legitimate

legitimate
An execution example

initial config.

strictly legitimate

legitimate

legitimate
Our results on tree orientation

- **Strongly stabilizing tree orientation protocol** resilient to a single Byzantine faults
  - distributed daemon
  - *C-radius*: 0 optimal
  - *C-times*: 1 optimal
    - Each process changes its parent at most once in transition from a legitimate config. to a strictly legitimate one.
  - **total disturbed time**: $O(n)$
    - The total amount of time violating problem specification is $O(n)$
Spanning tree construction

Assumption
- A single non-faulty root process exists
- All non-faulty processes form a connected component

Problem requirement
- Each process selects its parent so that non-faulty processes
  - form a rooted tree with the correct root, and/or
  - form rooted trees with the Byzantine root

Fault-free case

Faulty case
Spanning tree construction

Assumption
- A single non-faulty root process exists
- All non-faulty processes form a connected component

Problem requirement
- Each process selects its parent so that non-faulty processes
  - form a rooted tree with the correct root, and/or
  - form rooted trees with the Byzantine root

Fault-free case

Faulty case
Spanning tree construction

Assumption
- A single non-faulty root process exists
- All non-faulty processes form a connected component

Problem requirement
- Each process selects its parent so that non-faulty processes
  - form a rooted tree with the correct root, and/or
  - form rooted trees with the Byzantine root

Fault-free case

Faulty case
Spanning tree construction

Assumption
 A single non-faulty root process exists
 All non-faulty processes form a connected component

Problem requirement
 Each process selects its parent so that non-faulty processes
  • form a rooted tree with the correct root, and/or
  • form rooted trees with the Byzantine root
Our results on tree construction

- **Strong stabilization** for spanning tree construction
  - For any number of Malicious (yet correct players are connected)
    - Possibility of
      - $C$-radius = 0  optimal
      - $C$-times = $\Delta^d$  $\Delta$: max degree, $d$: diameter
Conclusion

- **Strict Stabilization**
  - Stabilization with *spatial* malicious containment

- **Strong stabilization**
  - Stabilization with *temporal* malicious containment

- **Strong stabilization for tree orientation** \( (n:\#\text{processes}) \)
  - **Impossibility** of \( C\)-times \( o(n) \) for two Malicious
  - A strongly stabilizing protocol for a single Malicious
    - \( C\)-radius = 0, \( C\)-times = 1 \( \text{optimal} \)

- **Strong stabilization for tree construction**
  - A strongly stabilizing protocol for any \#Malicious