A Synchronous View of Loosely Time-Triggered Architectures

Guillaume Baudart
Albert Benveniste
Timothy Bourke

Séminaire INFRES LINCS’15
Background

• **Quasi-synchrony:** Paul Caspi’s work on programming practices of Airbus engineers
  “no more than two ticks of one clock between two ticks of another one”
  [Caspi 2000, *Cooking book*]

• **LTTA:** Middleware to safely deploy synchronous applications over quasi-periodic architectures
  [Tripakis et al. 2008]
  [Caspi, Benveniste 2008]

• **Asynchrony:** Synchronous models of asynchronous systems
  [Halbwachs, Baghdadi 2002]
  [Halbwachs, Mandel 2006]
Outline

1. What is an LT TA?
   1. Quasi-Periodic Architecture
   2. Synchronous Applications

2. General Framework

3. The two protocols
   1. Back-Pressure LT TA
   2. Time-Based LT TA

4. What About Clock Synchronisation?
Outline

1. What is an LT TA?
   1. Quasi-Periodic Architecture
   2. Synchronous Applications

2. General Framework

3. The two protocols
   1. Back-Pressure LT TA
   2. Time-Based LT TA

4. What About Clock Synchronisation?
Quasi-Periodic Architecture

• A set of “quasi-periodic” processes with local clocks and nominal period $T^n$ (jitter $\varepsilon$)

\[
0 < T_{\text{min}} \leq T^n \leq T_{\text{max}} \quad \text{or} \quad T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon
\]

\[
(\kappa_i)_{i \in \mathbb{N}} \text{ clock activations}
\]

• Buffered communication without message inversion or loss

• Bounded communication delay

\[
\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}
\]
Quasi-Periodic Architecture

• A set of “quasi-periodic” processes with local clocks and nominal period $T^n$ (jitter $\varepsilon$)

\[ 0 < T_{\text{min}} \leq T^n \leq T_{\text{max}} \text{ or } T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon \]

\((\kappa_i)_{i \in \mathbb{N}}\) clock activations

• Buffered communication without message inversion or loss

• Bounded communication delay

\[ \tau_{\text{min}} \leq \tau \leq \tau_{\text{max}} \]
Quasi-Periodic Architecture

- A set of “quasi-periodic” processes with local clocks and nominal period $T^n$ (jitter $\varepsilon$)

$$0 < T_{\text{min}} \leq T^n \leq T_{\text{max}} \quad \text{or} \quad T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon$$

$(\kappa_i)_{i \in \mathbb{N}}$ clock activations

- Buffered communication without message inversion or loss

- Bounded communication delay
  $$\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$$
Problems

• **Overwriting:** Loss of values
• **Oversampling:** Duplication of values
• **Combination of signals**
Problems

- **Overwriting**: Loss of values
- **Oversampling**: Duplication of values
- **Combination of signals**
Problems

• **Overwriting:** Loss of values
• **Oversampling:** Duplication of values
• **Combination of signals**
Problems

• **Overwriting:** Loss of values
• **Oversampling:** Duplication of values
• **Combination of signals**

\[ a \land b \]

![Diagram showing combination of signals](image)
Problems

- **Overwriting**: Loss of values
- **Oversampling**: Duplication of values
- **Combination of signals**
Synchronous Applications
Network of Communicating Mealy Machines

- Initial state $S_{\text{init}}$
- Transition function $F : S \times \mathcal{V}^{n_i} \rightarrow S' \times \mathcal{V}^{n_o}$

Semantics

Synchronous $\llbracket m \rrbracket^S : (\mathcal{V}^{n_i})^\infty \rightarrow (\mathcal{V}^{n_o})^\infty$

Kahn $\llbracket m \rrbracket^K : (\mathcal{V}^\infty)^{n_i} \rightarrow (\mathcal{V}^\infty)^{n_o}$
Synchronous Applications
Network of Communicating Mealy Machines

• **Composition:** output to input

• **Causality:** no instantaneous dependency cycles

  *Basically, classic synchronous programs ‘mono-clock’.*

• **Assumption:** no instantaneous dependencies between nodes (avoid ‘microschedule’)*
What are LTTAs?

• **Base:** A quasi-periodic architecture

• **Goal:** Safely deploy a synchronous application

• **Idea:** Add a layer of middleware
Different Approaches

- **Discrete abstractions**, e.g., quasi-synchrony [Caspi 2000]
  - Allows verifications
  - State explosion, incomplete model

- **Petri nets**, [Benveniste et al. 2010]
  - Unify LTSA protocols
  - Complex model, no implementation

- **Zélus:**
  - A single language for discrete- and real-time,
  - Implementation, simulation using numerical solvers
  - But no verification at this point
Outline

1. What is an LT TA?
   1. Quasi-Periodic Architecture
   2. Synchronous Applications

2. General Framework

3. The two protocols
   1. Back-Pressure LT TA
   2. Time-Based LT TA

4. What About Clock Synchronisation?
General Framework
Modelling nodes
General Framework
Modelling nodes
General Framework
Modelling nodes
General Framework
Modelling nodes

Control the execution of the application
Logical clock model node activation
Synchronous application
General Framework
Modelling nodes

Control the execution of the application
Input sampled from memories (links)

Logical clock model node activation

Synchronous application

LTTA Controller
Application Machine

Input sampled from memories (links)

Control the execution of the application
General Framework
Modelling nodes

Control the execution of the application

Logical clock model node activation

Input sampled from memories (links)

Communication via signals $(v, b)$

Synchronous application

LTTA Controller

Application Machine
Control the execution of the application

Input sampled from memories (links)

Memory store the last computed value

Logical clock model node activation

Communication via signals (v, b)

Synchronous application

General Framework
Modelling nodes

LT TA Controller

Application Machine

Input sampled from memories (links)

Memory store the last computed value

Control the execution of the application

Logical clock model node activation

Communication via signals (v, b)

Synchronous application
General Framework
Modelling Links
General Framework
Modelling Links

Input: Signals
Input: Signals

Channel:
delay a signal

General Framework
Modelling Links
General Framework
Modelling Links

Input: Signals

Channel:
delay a signal

Logical clock:
model the transmission delay
General Framework
Modelling Links

Input: Signals

Channel:
delay a signal

Memory:
store the last received value

Logical clock:
model the transmission delay
What’s next?

Design controllers that ensure a synchronous execution of embedded machines

• **Back-Pressure LTTA**  
  [Tripakis et al. 2008]

• **Time-Based LTTA**  
  [Caspi, Benveniste 2008]
Outline

1. What is an LT TA?
   1. Quasi-Periodic Architecture
   2. Synchronous Applications

2. General Framework

3. The two protocols
   1. Back-Pressure LT TA
   2. Time-Based LT TA

4. What About Clock Synchronisation?
Back-Pressure Kahn Network

Buffer of Size 1

- Reading from a buffer is acknowledged to the writer
- Nodes alternate between **exec** and **send**
Back-Pressure Kahn Network

Buffer of Size 1

• Reading from a buffer is acknowledged to the writer
• Nodes alternate between exec and send
Back-Pressure Kahn Network

Buffer of Size 1

- Reading from a buffer is acknowledged to the writer
- Nodes alternate between `exec` and `send`
Back-Pressure LTTA

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism
Back-Pressure LT TA

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism
Back-Pressure LTTP

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism
Back-Pressure LTTA

- **Difference**: nodes are triggered by their local clock
- **Idea**: adding skipping mechanism
Back-Pressure LT TA

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism
Back-Pressure LT TA

forall_fresh(i) / emit im(i), a
forall_fresh(ra) / emit o(m)
Back-Pressure LTTA

- **Theorem 1:** Composition of the controller and the embedded machine is always well-defined (no cycle)

- **Theorem 2:** Back-pressure LTTA preserves the Kahn semantics of the embedded application (forget the skips)

- **Theorem 3:** The worst case throughput is: $1/\lambda_{BP} = 2(T_{\text{max}} + \tau_{\text{max}})$
Time-Based LTTA

- **Problem:** Back-pressure multiplies the number of messages and memories, and *blocks if a node crashes*

- **Idea:** Replace back-pressure by waiting, using timing characteristics of the architecture

- **First solution:** [Caspi, Benveniste 2008] Slow down the nodes to mimic a synchronous architecture, global synchronisation

- **Our proposal:** Relax broadcast assumption, localise synchronisations
Nodes alternate between **exec** and **send**

Sender sees publication of all receivers

**Idea:** At some point, a node can be sure that:
- the last sent data has been read
- a fresh value is available in the memory

**Time-Based LTTA**
Time-Based LTTPA

Wait
\( n = p \ fby \ n - 1 \)

Ready
\( n = q \ fby \ n - 1 \)

last \( n = 1 \) / emit \( \text{im}(i) \)

last \( n = 1 \) or exists_fresh(i) or exists_fresh(ro)

/ emit \( \text{o}(m) \)
\[
\text{Wait} \quad n = p \quad \text{fby} \quad n - 1 \\
\text{Ready} \quad n = q \quad \text{fby} \quad n - 1
\]

\[
\text{last } n = 1 \quad \text{or} \quad \text{exists}_\text{fresh}(i) \quad \text{or} \quad \text{exists}_\text{fresh}(ro) \\
/ \quad \text{emit} \quad o(m), \quad n = p
\]
Wait

Ready

Send

Receiver

last n = 1 / emit im(i), n = q

n = p fby n−1

n = q fby n−1

last n = 1 or exists_fresh(i) or exists_fresh(ro)

/ emit o(m), n = p

exec

send

exec

send
Wait

Ready

_sender_ 

Send

Receiver

last \( n = 1 \) or \( \text{exists}_{\text{fresh}}(i) \) or \( \text{exists}_{\text{fresh}}(\text{ro}) \) 

/ emit \( o(m), \ n = p \)
Ready

Wait

send

Last \( n = 1 \) / emit im(i), \( n = q \)

Wait

n = p fby n−1

Ready

n = q fby n−1

Last \( n = 1 \) or exists_fresh(i) or exists_fresh(ro)

/ emit o(m), \( n = p \)

Sender

Receiver

exec

send

3

2

0/3

2
sender execution

receiver execution

sender

receiver

"Wait" state

"Ready" state

"Send" state

"Exec" transition

"Send" transition

"Last n = 1 / Emit im(i), n = q"

"Last n = 1 or exists_fresh(i) or exists_fresh(ro) / Emit o(m), n = p"

"Ready n = q fby n-1" state

"Wait n = p fby n-1" state

"TB-LTTA" box

"c" input

"i" input

"ro" input

"m" input

"o" output

"im" output

"t" time

"n = 1" condition

"Send" action

"Exec" action

"Send" state

"Wait" state

"Ready" state

"Send" transition

"Exec" transition

"Send" transition

"Send" transition

"Send" transition

"Send" transition
Ready

Wait

23

Send

Receiver

send

last n = 1 / emit im(i), n = q

n = p fby n−1

Wait

n = q fby n−1

Ready

last n = 1 or exists_fresh(i) or exists_fresh(ro)

/ emit o(m), n = p

Sender

Receiver
\[
\text{Wait } n = p \text{ fby } n - 1 \\
\text{Ready } n = q \text{ fby } n - 1
\]

\[
\text{last } n = 1 \text{ or } \text{exists_fresh}(i) \text{ or } \text{exists_fresh}(ro) / \text{emit } o(m), n = p
\]

\[
\text{last } n = 1 / \text{emit } im(i), n = q
\]
sender

ready

wait

exec

send

3

2

1

0/2

receiver

exec

send

0/3

2

1

0/2
```
last n = 1 / emit im(i), n = q

Wait
n = p fby n−1

Ready
n = q fby n−1

last n = 1 or exists_fresh(i) or exists_fresh(ro)
/ emit o(m), n = p
```

Sender

```
<table>
<thead>
<tr>
<th>exec</th>
<th>send</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>exec</td>
<td>0/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
```

Receiver

```
<table>
<thead>
<tr>
<th>exec</th>
<th>send</th>
<th>0/3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>exec</td>
<td>0/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
\[
\text{last } n = 1 \text{ or exists}_\text{fresh}(i) \text{ or exists}_\text{fresh}(ro) \text{ or exists}_\text{fresh}(im) \\
/ \text{ emit } \text{o}(m), n = p
\]

\[
\text{last } n = 1 / \text{ emit } \text{im}(i), n = q
\]
Wait \rightarrow \text{Ready} \quad \text{send}

\begin{align*}
\text{Wait} \\
&\quad n = p \, \text{fb} \, n - 1 \\
\text{Ready} \\
&\quad n = q \, \text{fb} \, n - 1
\end{align*}

\text{last} \, n = 1 \quad \text{or} \quad \text{exists}_\text{fresh}(i) \quad \text{or} \quad \text{exists}_\text{fresh}(ro) \\
/ \, \text{emit} \, o(m), \, n = p

\text{Sender}

\text{Receiver}
Ready
Wait
23
exec
0/2
1
send
3
send
0/3
2

Receiver

Sender

Wait
Ready
Send
exec
send
3
2
1
exec
0/2
1
send
0/3
2

Wait
Ready
Send
exec
send
0/3
2
1
exec
0/2
3

last \( n = 1 \) or exists\_fresh(i) or exists\_fresh(ro) \\
/ emit \( o(m) \), \( n = p \)
Wait

i

ro

m

Send

Wait

Ready

exec

send

3

2

1

exec

0/2

1

send

0/3

2

Receiver

Wait

Ready

exec

send

0/3

2

1

exec

0/2

3
Time-Based LT TA

- **Theorem 1:** Composition of the controller and the embedded machine is always well-defined (no cycle)

- **Theorem 2:** Time-based LT TA preserves the Kahn semantics of the embedded application

- **Theorem 3:** The worst case throughput is: $1/\lambda_{TB} = (p + q)T_{\text{max}}$
The Time-Based Protocol

**Theorem 2:** The following initial counter values ensure the preservation of the Kahn semantics

\[
p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p
\]
The Time-Based Protocol

Proof sketch

- Worst case reasoning
- Tuning constants $p$ and $q$ (counter initial values)
- Ensure that the receiver always reads the proper data
The Time-Based Protocol

Property 1: $W^a_{k-1} < E^b_k$

$$p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}$$

$$q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left(\frac{T_{\text{max}}}{T_{\text{min}}} - 1\right)$$
The Time-Based Protocol

Property 1: \( W^a_{k-1} < E^b_k \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

Property 1: $W_{k-1}^a < E_k^b$

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p\left(\frac{T_{\text{max}}}{T_{\text{min}}} - 1\right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a < E_k^b \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p\left(\frac{T_{\text{max}}}{T_{\text{min}}} - 1\right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a < E_b^k \)

\[
p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}} \quad q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

Property 1: \( W^a_{k-1} \prec E^b_k \)

Property 2: \( E^b_k \prec W^a_k \)

\[
p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]
\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

Property 1: \( W^a_{k-1} \prec E^b_k \)

Property 2: \( E^b_k \prec W^a_k \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a < E_k^b \)

Property 2: \( E_k^b < W_k^a \)

\[
p > \frac{2\tau_{\max}}{T_{\min}} + \frac{T_{\max}}{T_{\min}}
q > \frac{\tau_{\max} - \tau_{\min}}{T_{\min}} + \frac{T_{\max}}{T_{\min}} + p \left( \frac{T_{\max}}{T_{\min}} - 1 \right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a \prec E_k^b \)

Property 2: \( E_k^b \prec W_k^a \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left(\frac{T_{\text{max}}}{T_{\text{min}}} - 1\right)
\]
Property 1: $W_{k-1}^a < E_k^b$

Property 2: $E_k^b < W_k^a$

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left(\frac{T_{\text{max}}}{T_{\text{min}}} - 1\right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a < E_k^b \)

Property 2: \( E_k^b < W_k^a \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}}
\]

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

Property 1: \( W_{k-1}^a < E_k^b \)

Property 2: \( E_k^b < W_k^a \)

\[
p > \frac{2\tau_{\text{max}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} \\
q > \frac{\tau_{\text{max}} - \tau_{\text{min}}}{T_{\text{min}}} + \frac{T_{\text{max}}}{T_{\text{min}}} + p \left( \frac{T_{\text{max}}}{T_{\text{min}}} - 1 \right)
\]
The Time-Based Protocol

**Corollary:** The protocol ensures alternation between **exec** and **send** phases for each pair of communicating nodes.

**Broadcast communication:** ensure clean alternation throughout the entire architecture (idem for back-pressure LT TA)
Comparison

Back-pressure

Time-based
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Time-Based</th>
<th>Back-Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flexibility</strong></td>
<td>Require architecture specifications</td>
<td>Very flexible</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>Can run in a degraded mode</td>
<td>Stuck if a node crash</td>
</tr>
<tr>
<td><strong>Fault Tolerance</strong></td>
<td>Can be programmed in the application</td>
<td>Implemented in the Middleware</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Pipelining</strong></td>
<td>Limited</td>
<td>Optimal</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Time-Based</th>
<th>Back-Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flexibility</strong></td>
<td>Require architecture specifications</td>
<td>Very flexible</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td>Can run in a degraded mode</td>
<td>Stuck if a node crash</td>
</tr>
<tr>
<td><strong>Fault Tolerance</strong></td>
<td>Can be programmed in the application</td>
<td>Implemented in the Middleware</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Pipelining</strong></td>
<td>Limited</td>
<td>Optimal</td>
</tr>
</tbody>
</table>
Outline

1. What is an LT TA?
   1. Quasi-Periodic Architecture
   2. Synchronous Applications

2. General Framework

3. The two protocols
   1. Back-Pressure LT TA
   2. Time-Based LT TA

4. What About Clock Synchronisation?
Global Clock Protocol

Central Master Synchronisation

- **Goal:** Implement clock synchronisation on a quasi-periodic architecture
- We use the most efficient protocol for comparison purposes
- One node is used as a time reference for all other nodes: the **central master clock**
Global Clock Protocol

The Big Picture

Reference time

Synchronisation interval

Maximum divergence

Reference time (Master)

Offset after resynchronisation

Local time
Global Clock Protocol

Central Master Synchronisation

Synchronisation interval

Offset after resynchronisation

Drift rate

Maximum divergence

Precision

\[ R \]

\[ \Phi = \tau_{\text{max}} + T_{\text{max}} - \tau_{\text{min}} \]

\[ \rho = \frac{T_{\text{max}}}{T_n} - 1 = \frac{T_{\text{max}} - T_{\text{min}}}{T_{\text{max}} + T_{\text{min}}} \]

\[ \Gamma = 2\rho R \]

\[ \Pi = \Phi + \Gamma \]
Global Clock Protocol

Lock-Step Execution

• Given the precision of the synchronisation one can build a global notion of time or macrosteps.

• A lock-step execution can be achieved if nodes execute once over 4 macrosteps [Kopetz 1997].

• We also need to wait for the transmission delay between execution steps (buffers of size 1).
Overhead Comparison
Compared to synchronous execution*

Node: $10^{-2}s$  Transmission: $10^{-7}s$

* The smaller, the better.
Overhead Comparison

Compared to synchronous execution*

Node: $10^{-4}s$  
Transmission: $10^{-4}s$

* The smaller, the better.
Overhead Comparison

Compared to synchronous execution*

Node: $10^{-7}s$  Transmission: $10^{-2}s$

* The smaller, the better.
Conclusion

• Our new model simplifies and clarifies those of previous papers

• A new proposition for the time-based protocol that does not require broadcast communication and does allow pipelining

• Model and Simulation of the protocols in Zélus Discrete model + link with continuous time

• Comparison with clock synchronisation deployed on the same architecture