Loosely Time-Triggered Architectures: Improvements and Comparisons

Guillaume Baudart
Albert Benveniste
Timothy Bourke

EMSOFT’15

Amsterdam, 06-10-2015
Each machine is characterized by
- an initial state \( S_{\text{init}} \)
- a transition function
\[
F : S \times V^I \rightarrow S \times V^O
\]
Machines communicate through unit delays.
Synchronous Applications...
Composition of Communicating Mealy Machines

• Each machine is characterized by
  - an initial state $S_{init}$
  - a transition function
    $$F : S \times V^I \rightarrow S \times V^O$$

• Machines communicate through unit delays.

**Composition** forms a global synchronous application.
Synchronous Applications...
Composition of Communicating Mealy Machines

- Each machine is characterized by
  - an initial state $S_{\text{init}}$
  - a transition function
    \[ F : S \times V^I \rightarrow S \times V^O \]

- Machines communicate through unit delays.

Composition forms a global synchronous application.

Semantics: Sequence of values on each variable
...on 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

$$\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$$
...on 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

$$\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$$
...on 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

$$\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$$
Sampling Artifacts

• **Overwriting**: Loss of values

• **Oversampling**: Duplication of values

• **Combination of signals**

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
Sampling Artifacts

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

Example from [Caspi 2000]
How to Preserve the Semantics?
(of a synchronous application on a quasi-periodic architecture)
How to Preserve the Semantics?
(of a synchronous application on a quasi-periodic architecture)

Clock synchronization

e.g. TTA [Kopetz, Bauer 2003]

Now efficient and cheap.
How to Preserve the Semantics?
(of a synchronous application on a quasi-periodic architecture)

Clock synchronization
e.g. TTA [Kopetz, Bauer 2003]

Now efficient and cheap.

Unsynchronized nodes
+ Middleware = LTTA

Are they a good idea?
A Synchronous Framework

Everything can be expressed as a Zélus program
- discrete control (application and controllers)
- continuous time (architecture)

A Synchronous Framework

Everything can be expressed as a Zélus program
- discrete control (application and controllers)
- continuous time (architecture)


[Halbwachs, Baghdadi 2002]
A Synchronous Framework

Everything can be expressed as a **Zélus** program
- discrete control (application and controllers)
- continuous time (architecture)


[Halbwachs, Baghdadi 2002]
A Synchronous Framework

Everything can be expressed as a **Zélus** program
- discrete control (application and controllers)
- continuous time (architecture)

Controls the execution of the application

Logical clock models node activation

Synchronous application


[Halbwachs, Baghdadi 2002]
A Synchronous Framework

Everything can be expressed as a Zélus program
- discrete control (application and controllers)
- continuous time (architecture)

Controller waits for new inputs and delays publications
A Synchronous Framework

Everything can be expressed as a Zélus program
- discrete control (application and controllers)
- continuous time (architecture)

Controller \textbf{waits} for new inputs and \textbf{delays} publications

Controls the execution of the application

Input sampled from memories (links)

Logical clock models node activation

Synchronous application


[Halbwachs, Baghdadi 2002]
A Synchronous Framework

Everything can be expressed as a **Zélus** program

- discrete control (application and controllers)
- continuous time (architecture)

Controller **waits** for new inputs and **delays** publications
A Synchronous Framework

Everything can be expressed as a Zélus program
- discrete control (application and controllers)
- continuous time (architecture)

Controller waits for new inputs and delays publications
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.

[Diagram showing M1 sending a message to M2 with acknowledgements]

[Tripakis et al. 2008]
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

- Nodes **skip** when no acknowledgement or message has been received.
A First Idea: Back-Pressure

• A producer waits for **acknowledgements** from its consumers before sending a new value.

• Nodes **skip** when no acknowledgement or message has been received.
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.
A First Idea: Back-Pressure

- A producer waits for acknowledgements from its consumers before sending a new value.
- Nodes skip when no acknowledgement or message has been received.
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.

- Nodes **skip** when no acknowledgement or message has been received.
A First Idea: Back-Pressure

- A producer waits for **acknowledgements** from its consumers before sending a new value.
- Nodes **skip** when no acknowledgement or message has been received.

**Flexibility:** there is no assumption on the architecture.

**Robustness:** if a node crashes the entire network is stuck.
Controller **waits** for new inputs and **delays** publications

---

**A First Idea: Back-Pressure**

Controller waits for new inputs and delays publications.
A First Idea: Back-Pressure

Controller \textbf{waits} for new inputs and \textbf{delays} publications
A First Idea: Back-Pressure

Controller **waits** for new inputs and **delays** publications
A First Idea: Back-Pressure

Controller **waits** for new inputs and **delays** publications.
A First Idea: Back-Pressure

Controller **waits** for new inputs and **delays** publications
A First Idea: Back-Pressure

Controller *waits* for new inputs and *delays* publications.
A First Idea: Back-Pressure

Controller **waits** for new inputs and **delays** publications.
A First Idea: Back-Pressure

Controller **waits** for new inputs and **delays** publications
Controller **waits** for new inputs and **delays** publications.
A First Idea: Back-Pressure

Controller \textbf{waits} for new inputs and \textbf{delays} publications.
A First Idea: Back-Pressure

Controller \textbf{waits} for new inputs and \textbf{delays} publications.
Time-Based LTTA

Why another protocol?

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

We can take advantage of the **quasi-periodic** nature of the architecture to replace acknowledgment by **waiting**.

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 LT TA

Why another protocol?

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

We can take advantage of the **quasi-periodic** nature of the architecture to replace acknowledgment by **waiting**.

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

**Flexibility:** it requires architecture characteristics.
**Robustness:** controllers can run in a degraded mode.
**Time-Based LTTA**

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.
Time-Based LTTA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.
### Time-Based LTTA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

---

**Diagram Details:**

- **Init:** $n = 1$
- **Last:** $n = 1$ / emit $im = data(i)$
- **Wait:** $n = p \rightarrow (last \ n - 1)$
- **Ready:** $n = q \rightarrow (last \ n - 1)$
- **Countdown:** $last \ n = 1$ or preempted / emit $o = m$

---

**TB-LTTA**
Time-Based LTTA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

Input for the application
**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.
Time-Based LTTA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

- **Wait:** \( n = p \rightarrow (\text{last } n - 1) \)
  - \( \text{last } n = 1 \) or preempted / \( \text{emit } o = m \)
- **Ready:** \( n = q \rightarrow (\text{last } n - 1) \)
  - \( \text{init } n = 1 \)
  - \( \text{last } n = 1 \) / \( \text{emit } im = \text{data}(i) \)

Countdown

- **Output of the application**
- **Input for the application**
Time-Based LT TA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

\[
\text{init } n = 1 \\
\text{last } n = 1 \quad / \quad \text{emit } im = \text{data}(i)
\]

\[
\text{last } n = 1 \quad / \quad \text{emit } o = m
\]
Time-Based LT TA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

```
init n = 1
last n = 1 / emit im = data(i)
```

```
last n = 1 or preempted / emit o = m
```
Time-Based LT TA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

**Preemption:** when a publication is detected, the consumers have finished executing.
Time-Based LTTA

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

**Preemption:** when a publication is detected, the consumers have finished executing.
**Time-Based LTTA**

**Wait:** await the publication of the slowest node.

**Ready:** resynchronize with the fastest node.

**Preemption:** when a publication is detected, the consumers have finished executing.
Time-Based LT TA
Simulation $p=3$ $q=2$

```

ten = 1
last n = 1 / emit im = data(i)

Wait
n=p→(pre n-1)

Ready
n=q→(pre n-1)

last n = 1 or preempted / emit o = m
```

Send

Exec
Time-Based LTTA
Simulation $p=3$ $q=2$

\begin{align*}
\text{Wait} & \\
\text{Ready} & \\
\end{align*}
Time-Based LTTA
Simulation $p=3$ $q=2$

init $n = 1$

last $n = 1$ / emit $im = data(i)$

$\text{Wait } n = p \rightarrow (\text{pre } n - 1)$

$\text{Ready } n = q \rightarrow (\text{pre } n - 1)$

last $n = 1$ or preempted / emit $o = m$

Send $\rightarrow$ Exec

$\text{Wait } \xrightarrow{\text{exec}} \text{Ready}$

$\text{Wait } \xrightarrow{\text{send}} \text{Ready}$

Send $\rightarrow$ Exec

$\text{Wait } \xrightarrow{\text{exec}} \text{Ready}$

$\text{Wait } \xrightarrow{\text{send}} \text{Ready}$

$t$

$t$

$0/3$

$3$
Time-Based LTTA
Simulation p=3 q=2
Time-Based LTTA
Simulation p=3 q=2
Time-Based LTTA
Simulation p=3 q=2

$$\text{init } n = 1$$
$$\text{last } n = 1 \rightarrow \text{emit } im = \text{data}(i)$$

$$\text{Wait } n=p \rightarrow (\text{pre } n-1)$$

$$\text{Ready } n=q \rightarrow (\text{pre } n-1)$$

$$\text{last } n = 1 \text{ or preempted } \rightarrow \text{emit } o = m$$
Time-Based LTTA

Simulation $p=3$ $q=2$

![Diagram of Time-Based LTTA with states and transitions]

- **Send**
- **Exec**

**TB-LTTA**

- **Init**: $n = 1$
- **Last**: $n = 1$ / emit $im = data(i)$
- **Wait**: $n = p \rightarrow (pre\ n-1)$
- **Ready**: $n = q \rightarrow (pre\ n-1)$
- **Last**: $n = 1$ or preempted / emit $o = m$

**States and Transitions**:

- **Wait**
- **Ready**

**Time Line**:

- $t = 0/3$
- $t = 2$
- $t = 3$
- $t = 2/0$

**Graphical Representation**:

- Nodes and edges representing the transitions and states in the Time-Based LTTA simulation.
Time-Based LTTA
Simulation p=3 q=2

**Send**

**Exec**

```
init n = 1
last n = 1 / emit im = data(i)
```

```
Wait
n=p→(pre n−1)
```

```
Ready
n=q→(pre n−1)
```

```
last n = 1 or preempted / emit o = m
```
Time-Based LT TA

Simulation $p=3$ $q=2$

- **Send**
- **Exec**

Diagram:

- **Wait**
  - Send
  - Exec

- **Ready**
  - Exec

- TB-LTTA
  - Init $n=1$
  - Last $n=1$ / Emit $im = data(i)$
  - Ready $n=q\rightarrow(pre\ n-1)$
  - Last $n=1$ or preempted / Emit $o=m$

Timeline:

- Time $t$

Graph:

- Nodes: 3, 2, 1, 0/2, 1
- Edges:
  - From 3 to 2
  - From 2 to 1
  - From 1 to 0/2
  - From 0/2 to 1

Legend:

- **Send**
- **Exec**
Time-Based LTTA

Simulation p=3 q=2
Time-Based LTTA
Simulation $p=3$ $q=2$

init $n = 1$

\[ \text{last } n = 1 \rightarrow \text{emit } im = \text{data}(i) \]

\[ n=p \rightarrow (\text{pre } n-1) \]

\[ n=q \rightarrow (\text{pre } n-1) \]

\[ \text{last } n = 1 \text{ or preempted } \rightarrow \text{emit } o = m \]
Time-Based LT TA

Simulation $p=3$ $q=2$

- **Send**: Blue circles
- **Exec**: Red circles

### Time-Based LT TA (TB-LTTA)

- **Init**: $n = 1$
- **Last**: $n = 1$ / emit $i = \text{data}(i)$
- **Wait**: $n=p\to\text{pre}\ n-1$
- **Ready**: $n=q\to\text{pre}\ n-1$
- **Last**: $n = 1$ or preempted / emit $o = m$

---

**Graphs**

- **Nodes**: 3, 2, 1, 0/2, 1, 0/3
- **Links**: Send and Exec transitions
- **Time**: $t$
Time-Based LTTA
Simulation p=3 q=2

Nodes alternate between **send** and **exec** phases
Theorem 1:
The composition of the controller and the application is always well-defined (no causality cycle).

Theorem 2:
The following constraints on the initial counter values ensure the preservation of the semantics

\[ p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}} \]
\[ q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p \]

Theorem 3:
The worst case throughput is given by
\[ \frac{1}{\lambda_{\text{TB}}} = (p + q)T_{\text{max}} \]
Preservation of the Semantics

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

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

\[
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
Preservation of the Semantics

Property 1: $S_{k-1}^a \prec E_k^b$

$p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}}$
$q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p$
Preservation of the Semantics

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

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

\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}}
\]
Preservation of the Semantics

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

$$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$$
$$q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p$$
Preservation of the Semantics

Property 1: $S_{k-1}^a \prec E_k^b$

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

$$q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p$$
Preservation of the Semantics

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

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

\[
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
Preservation of the Semantics

Property 1: $S_{k-1}^a \prec E_k^b$

$p > \frac{2\tau_{max} + T_{max}}{T_{min}}$
$q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p$
Preservation of the Semantics

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

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

\[
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
Preservation of the Semantics

\[
\begin{align*}
p & > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}} \\
q & > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p
\end{align*}
\]

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

Property 2: \( E_k^b < S_k^a \)
Preservation of the Semantics

Property 1: \( S_{k-1}^a < E_k^b \)
Property 2: \( E_k^b < S_k^a \)

\[
p > \frac{2\tau_{max} + T_{max}}{T_{min}}
\]
\[
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
Preservation of the Semantics

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

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

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

\[
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
Preservation of the Semantics

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

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

\[
p > \frac{2\tau_{max} + T_{max}}{T_{min}}
q > \frac{\tau_{max} - \tau_{min} + (p + 1)T_{max}}{T_{min}} - p
\]
**Preservation of the Semantics**

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

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

\[
p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}}
q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p
\]
Preservation of the Semantics

Property 1: $S_{k-1}^a \prec E_k^b$

Property 2: $E_k^b \prec S_k^a$

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

$$q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p$$
Preservation of the Semantics

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

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

\[
p > \frac{2\tau_{\text{max}} + T_{\text{max}}}{T_{\text{min}}} - p
\]
\[
q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p
\]
Preservation of the Semantics

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

Property 2: \( E^b_k < S^a_k \)

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

\[ q > \frac{\tau_{\text{max}} - \tau_{\text{min}} + (p + 1)T_{\text{max}}}{T_{\text{min}}} - p \]
What about Clock Synchronization?
Worst-Case Evaluation
Analytical comparison with synchronous execution*

Node : $10^{-2}$s  
Transmission: $10^{-6}$s

- Back-Pressure
- Time-Based
- Global-Clock

Slowdown

0x  2x  4x  6x  8x

0%  7,5%  15%  22,5%  30%

Jitter

*The smaller, the better
Worst-Case Evaluation
Analytical comparison with synchronous execution*

Node: $10^{-2}$s
Transmission: $10^{-6}$s

- Back-Pressure
- Time-Based
- Global-Clock

However:
- Global-Clock is as efficient as possible
- LTTA are **simpler** protocols (two control states)
- Time-Based is the **least intrusive**

*The smaller, the better
Worst-Case Evaluation
Analytical comparison with synchronous execution*

Node: $10^{-6}s$  
Transmission: $10^{-2}s$

- Back-Pressure
- Time-Based
- Global-Clock

Slowdown

0x 2,5x 5x 7,5x 10x

0% 7,5% 15% 22,5% 30% Jitter

However:
- Global-Clock is as efficient as possible
- LTTA are **simpler** protocols (two control states)
- Time-Based is the **least intrusive**

*The smaller, the better*
Contributions

• Clarification of the assumptions on the synchronous application

• A unifying synchronous framework for LTTAs that gives executable code for simulation

• Simplification of the Time-Based protocol
  - Relaxing broadcast communication
  - No more global synchronization

• Theoretical comparison with clock synchronization deployed on the same architecture
Contributions

• Clarification of the assumptions on the synchronous application

• A unifying synchronous framework for LTTAs that gives executable code for simulation

• Simplification of the Time-Based protocol
  - Relaxing broadcast communication
  - No more global synchronization

• Theoretical comparison with clock synchronization deployed on the same architecture

Erratum