Synchronous Modeling of
Loosely Time-Triggered Architectures

Guillaume Baudart
Albert Benveniste

with many thanks to Timothy Bourke, Adrien Guatto and Marc Pouzet
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 are LTTAs?
2. Synchronous model
3. The two protocols
4. Comparison
Synchronous Applications
Network of communicating Mealy Machines

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

Semantics

Synchronous \( \llbracket m \rrbracket^S : (V^{n_i})^\infty \rightarrow (V^{n_o})^\infty \)
Kahn \( \llbracket m \rrbracket^K : (V^\infty)^{n_i} \rightarrow (V^\infty)^{n_o} \)
Synchronous Applications

Network of communicating Mealy Machines

• **Composition**: output to input

• **Causality**: no instantaneous dependency cycles

  Basically, classic synchronous programs without clocks.

**Example**

```plaintext
let node from m = nat where
  rec nat = m -> pre nat + 1
```
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
  
  $$\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}$$
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

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

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

\[
0 < T_{\text{min}} \leq T^n \leq T_{\text{max}} \quad \text{or}
\]

\[
T^n - \varepsilon \leq \kappa_i - \kappa_{i-1} \leq T^n + \varepsilon
\]

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

\[
\tau_{\text{min}} \leq \tau \leq \tau_{\text{max}}
\]
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

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

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

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

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

\[ a_t \land b_t \]
What are LTTAs?

- **Base:** A quasi-periodic architecture
- **Goal:** Safely deploy a synchronous application
- **Idea:** Add a layer of middleware
Synchronous Model

Modeling the nodes
Synchronous Model

Modeling the nodes
Synchronous Model

Modeling the nodes

Logical clock model activation

Synchronous application
Synchronous Model

Modeling the nodes

Control the execution of the application

Logical clock model activation

Shell-Wrapper

Controller

Embedded Machine

Synchronous application
Synchronous Model
Modeling the nodes

Control the execution of the application
Logical clock model activation

Input sampled from a memory
Synchronous application

Control the execution of the application
Logical clock model activation

Input sampled from a memory
Synchronous application
Synchronous Model

Modeling the nodes

Control the execution of the application

Logical clock model activation

Input sampled from a memory

Synchronous application

Communication via signals (v, b)
Synchronous Model
Modeling the nodes

Input sampled from a memory

Control the execution of the application

Logical clock model activation

Synchronous application

Communication via signals (v, b)

let node ltta_node (c, i) = o where
rec (o, im) = controller (c, i, om)
and present im(v) -> do emit om = em v done
Synchronous Model
Modeling the links
Synchronous Model
Modeling the links

Signal sent by a node
Transmission

Synchronous Model
Modeling the links

Channel: delay a signal

Signal sent by a node
Synchronous Model

Modeling the links

Channel: delay a signal

Logical clock: model the transmission delay

Signal sent by a node
Synchronous Model
Modeling the links

Channel: delay a signal

Logical clock: model the transmission delay

Signal sent by a node

Memory: store the last received value

Signal sent by a node
Synchronous Model

Modeling the links

Channel:
delay a signal

Logical clock:
model the transmission delay

let node channel (cl, i) = o where
  rec init mem = empty
  and present i(v) -> do mem = enqueue (last mem, v) done
  | cl  -> do emit o = front (last mem)
  and mem = dequeue (last mem) done

Signal sent
by a node

Memory:
store the last received value

let node mem (i, default) = m where
  rec init m = default
  and present i(v) -> do m = v done
Synchronous Model

Freshness of values

• **Problem:** Determine if a new value has arrived

• **Idea:** Add an *alternating bit protocol* to the channel
Synchronous Model
Freshness of values

• **Problem:** Determine if a new value has arrived

• **Idea:** Add an *alternating bit protocol* to the channel

```haskell
let node fresh i = o where
  rec init s = false
  and o = xor (last s, i.alt)
  and present o -> do s = i.alt done
```
Synchronous Model
From discrete to physical time

**Timing function:** $T : \mathcal{C} \rightarrow \mathbb{N} \rightarrow \mathbb{R}$
associate a time-tag to the $k^{th}$ activation
of a logical clock

**Node**

$$\forall i \in \mathbb{N} \quad T_{\text{min}} \leq T(c^n)(i + 1) - T(c^n)(i) \leq T_{\text{max}}$$

**Link**

$$\forall i \in \mathbb{N} \quad T(c^l)(i) = T(c^s)(i) + \tau_i$$

with $\tau_{\text{min}} \leq \tau_i \leq \tau_{\text{max}}$
Synchronous Model
From discrete to physical time

Node

```plaintext
let hybrid metro (t_min, t_max) = c where
  rec der t = 1.0 init . Misc.rand_val (t_min, t_max)
  reset c() -> . Misc.rand_val (t_min, t_max)
  and present up(last t) -> do emit c = () done
```

Link

```plaintext
let hybrid delay (c, tau_min, tau_max) = s where
  rec der t = 1.0 init 0.0
  reset c() -> . Misc.rand_val (tau_min, tau_max)
  and present up(last t) -> do emit s = () done
```
Synchronous Model
From discrete to physical time

• **Other approaches:** Discrete abstractions of the characteristics of the architecture, e.g., quasi-synchrony

• **Problem:** Does not model the transmission delay (modeled as one tick of the base clock)
  State explosion

• **ODE:** Easy simulation, directly relates to the architecture description
  But no verification…
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]
Back-Pressure Kahn Network

Buffer of size 1

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

Buffer of size 1

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

Buffer of size 1

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

Buffer of size 1

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

Buffer of size 1

- Reading from a buffer is acknowledged to the writer
- Nodes alternate between exec and write
Back-Pressure LTNA

- **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

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

- **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

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

- **Difference:** nodes are triggered by their local clock
- **Idea:** adding skipping mechanism
Back-Pressure LT TA
let node bp_controller (c, i, ar, om, default) = (a, o, im) where
  rec fi = fresh i
  and far = fresh ar
  and m = mem (om, default)
  and init state = Wait
  and match c with
    | false -> do done
    | true ->
      do match last state, fi, far with
        (* exec *) | Wait, true, _ ->
          do state = Ready
          and emit im = i.data
          and emit a = true done
        (* write *) | Ready, _, true ->
          do state = Wait
          and emit o = m done
        (* skip *)  | _ -> do done
          done
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:
  \[ \frac{1}{\lambda_{BP}} = 2(T_{\text{max}} + \tau_{\text{max}}) \]
Time-Based LT TA

• **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

• **Our proposal:** Relax broadcast assumption, localise synchronisations
Time-Based LTTA

- Nodes alternate between `exec` and `writes`.
- Sender sees publication of the receiver.
- **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 LT TA

TB-LTTA

WAIT

READY

Embedded Machine

\[ n := n - 1 \]

\[ (o^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write} \]

\[ (n = 0) / n := q, \text{ exec} \]

\[ n := n - 1 \]
Time-Based LTFA

```haskell
let node tb_controller (c, i, ro, om, default) = (o, im) where
  rec fi = fresh i
  and fro = fresh ro
  and init mem = default
  and init n = p
  and init state = Wait
  and match c with
  | false -> do done
  | true ->
    do
      match last state, (last n = 1), (fro or fi) with
      (* exec *) | Wait, true, _ ->
        do state = Ready and n = q and mem = om
        and emit im = i done
      (* write *) | Ready, _, true | Ready, true, _ ->
        do state = Wait and n = p
        and emit o = last mem done
      (* wait *) | _ -> do n = (last n) - 1 done
      done
```
(n = 0) / n := q, exec  

(n = 0) or (i) or (n = 0) / n := p, write 

 Sender

 Receiver
TB-LTTA

WAIT

\( n := n - 1 \)

READY

\( n := n - 1 \)

\( (\sigma^r)^* \) or \( (i)^* \) or \( (n = 0) / n := p, \ write \)

\( (n = 0) / n := q, \ exec \)

Sender

Receiver

write

write

\( 3 \)

\( 0/3 \)
TB-LTTA

Wait

$(n = 0) / n := q, \text{ exec}$

$(\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write}$

$n := n - 1$

$\text{ready}$

$\text{write}$

Sender

$\text{write}$

Receiver

$\text{write}$
TB-LTTA

\[ (n = 0) \rightarrow n := q, \text{ exec} \]

\[ n := n - 1 \rightarrow \]

\[ (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) \rightarrow n := p, \text{ write} \]

\[ n := n - 1 \rightarrow \]

\[ \text{write} \]

\[ \text{Sender} \]

\[ \text{Receiver} \]
TB-LTTA

\[(n = 0) \lor n := q, \text{ exec}\]

\[n := n - 1\]

\[(o^r)^* \lor (i)^* \lor (n = 0) \lor n := p, \text{ write}\]

\[n := n - 1\]

**Sender**

**Receiver**
TB-LTTA

\[
(n = 0) / \ n := q, \ \text{exec} \\
\]

\[
\begin{align*}
\text{WAIT} & \quad n := n - 1 \\
\text{READY} & \quad n := n - 1 \\
\end{align*}
\]

\[
(o^r)^* \text{ or } (i)^* \text{ or } (n = 0) / \ n := p, \ \text{write}
\]

**Sender**

**Receiver**

\[
\begin{align*}
\text{write} & \quad 3 \quad 2 \quad 1 \\
\text{write} & \quad 0/3 \quad 2 \quad 1
\end{align*}
\]
TB-LTTA

\[ (n = 0) \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write } \]

```
WAIT
n := n - 1
\rightarrow
REDA
n := n - 1
```

\[ (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := q, \text{ exec } \]

Sender

```
write
3
↑
Wait
↓
2
Read
↓
1
↓
0/2
```

Receiver

```
write
0/3
↑
Wait
↓
2
↓
1
```

\[ n := n - 1 \]

\[ n := n - 1 \]
TB-LTTA

\[\begin{align*}
\text{WAIT} & \quad (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write} \\
\text{READY} & \quad (n = 0) / n := q, \text{ exec} \\
\end{align*}\]

\[n := n - 1\]

**Sender**

\[\text{write} \quad 3 \quad 2 \quad 1 \quad \text{read} \quad 0/2\]

**Receiver**

\[\text{write} \quad 0/3 \quad 2 \quad 1 \quad \text{read} \quad 0/2\]
TB-LTTA

\[
(n = 0) / n := q, \text{ exec}
\]

\[
(n = 0) / n := p, \text{ write}
\]

\[
\begin{align*}
&n := n - 1 \\
&(\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := q, \text{ exec} \\
&n := n - 1
\end{align*}
\]

**Sender**

- **Wait**: 3
- **Ready**: 2
- **Read**: 1
- **Write**: 0/2

**Receiver**

- **Wait**: 0/3
- **Ready**: 2
- **Read**: 1
- **Write**: 0/2
TB-LTTA

\( (n = 0) / n := q, \text{ exec} \)

\( n := n - 1 \)

\( (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write} \)

\( n := n - 1 \)

Sender

write

3 2 1 0/2 1

Receiver

write

0/3 2 1 0/2 1

\[ n := n - 1 \]
TB-LTTA

\[ n := n - 1 \]

\[ (n = 0) / n := q, \text{ exec} \]

\[ (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write} \]

---

**Sender**

- **Write**: 3 → 2 → 1
- **Read**: 0/2 → 1
- **Write**: 0/3

**Receiver**

- **Write**: 0/3 → 2 → 1
- **Read**: 0/2 → 1
TB-LTTA

\[ (n = 0) \rightarrow n := q, \text{ exec} \]

\[ n := n - 1 \rightarrow \text{ WAIT} \]

\[ \text{write} \]

\[ (a^r)^* \text{ or } (i)^* \text{ or } (n = 0) \rightarrow n := p, \text{ write} \]

\[ n := n - 1 \rightarrow \text{ READY} \]

Sender

Receiver
TB-LTTA

\[
(n = 0) / n := q, \text{ exec}
\]

\[
\begin{align*}
 n := n - 1 \\
\end{align*}
\]

\[
(σ^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write}
\]

\[
\begin{align*}
 n := n + 1
\end{align*}
\]
TB-LTTA

\[ (n = 0) / n := q, \text{ exec} \]

\[ n := n - 1 \]

\[ (\sigma^r)^* \text{ or } (i)^* \text{ or } (n = 0) / n := p, \text{ write} \]

\[ n := n - 1 \]

StateMachine Diagram:

**Sender**

- Start in **Wait** state.
- Transition to **Ready** state with `exec`.
- Transition back to **Wait** state with `write`.

**Receiver**

- Start in **Wait** state.
- Transition to **Ready** state with `exec`.
- Transition back to **Wait** state with `write`.

Timeline:

- **Write**: 3, 2, 0/3, 2, 3
- **Read**: 1, 0/2, 1, 0/2, 1
- **Write**: 0/3, 2, 1, 1, 2

Note: The diagram includes state transitions and actions associated with `write` and `read` operations.
Time-Based LTAA

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

• **Theorem 2:**
  Time-based LTAA preserves the Kahn semantics of the embedded application

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

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

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

Proof sketch

• Worst case reasoning

• Tuning constant p and q (counter initial values)

• Ensure that the receiver always read the proper data
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: \( W_{k-1}^a < E_k^b \)

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

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

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

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

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

Property 2: $E^b_k < W^a_k$

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

Property 1: \[ W_{k-1}^a \prec E_k^b \]

Property 2: \[ E_k^b \prec W_k^a \]

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

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

Property 1: \[ W_{k-1}^a < E_k^b \]

Property 2: \[ E_k^b < W_k^a \]

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

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

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

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

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

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

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

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

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

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

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

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

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

**Corollary:** The protocol ensures alternation between exec and read 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>Pipeline</strong></td>
<td>Yes</td>
<td>Optimal</td>
</tr>
<tr>
<td><strong>Throughput (local)</strong></td>
<td>$1/4T_{\text{max}}$</td>
<td>$1/2T_{\text{max}}$</td>
</tr>
<tr>
<td><strong>Throughput (distant)</strong></td>
<td>$1/2\tau_{\text{max}}$</td>
<td>$1/2\tau_{\text{max}}$</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>Pipeline</strong></td>
<td>Yes</td>
<td>Optimal</td>
</tr>
<tr>
<td><strong>Throughput (local)</strong></td>
<td>$1/4T_{\text{max}}$</td>
<td>$1/2T_{\text{max}}$</td>
</tr>
<tr>
<td><strong>Throughput (distant)</strong></td>
<td>$1/2\tau_{\text{max}}$</td>
<td>$1/2\tau_{\text{max}}$</td>
</tr>
</tbody>
</table>
Conclusion

• Synchronous model of both the embedded application and the middleware

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

• Simulation of the protocol in Zélus Discrete model + link with continuous time
Next?

• Formal verification of the protocol. Problem: parametrised by the number of nodes

• Model the non-determinism of the architecture. Discrete abstraction (quasi-synchrony is not enough!)

• Real world experiments with LTTA protocols