# Asynchronous Middleware for Achieving Adaptivity and Dependability

Gérard Le Lann
INRIA
ADEPT Project

Gerard.Le_Lann@inria.fr

**1. Prolegomena (3 – 9)**

**2. On middleware and computational models (10 – 25)**

**3. Asynchronous real-time: the “late binding” principle (26 - 33)**

**4. Conclusions (34)**

# 1. Prolegomena

Application problems of interest? Those such that properties P required at M/W level are some combination of Safety (SafeP), Liveness (LiveP), Timeliness (TimeP), Dependability (DepP). Furthermore, "predictions before running" are mandatory.

Real (application) problem RΠ ⦊ Problem Π (computing systems, computer science/engineering, …) at M/W level

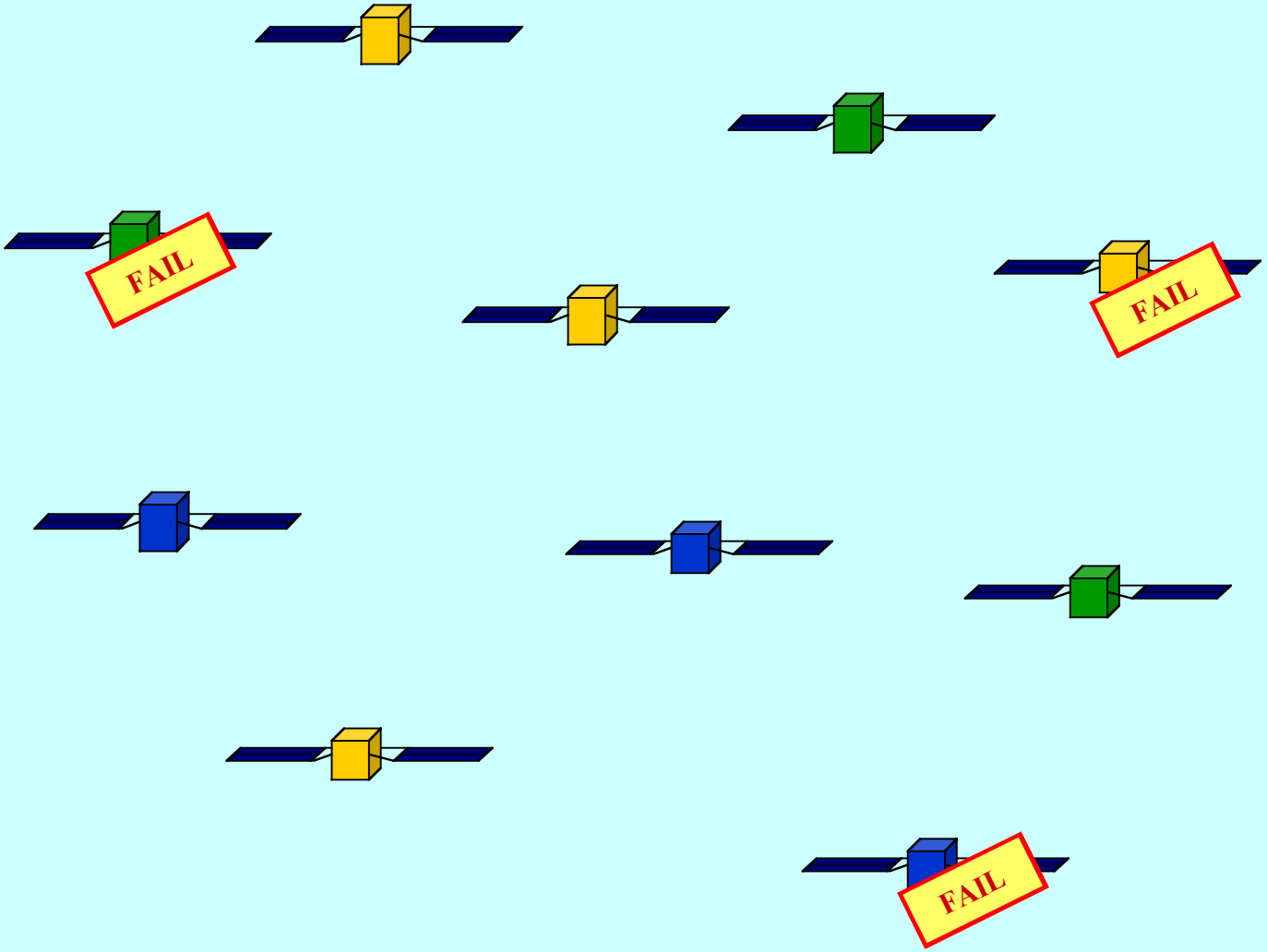
◊ **M/W design solution Δ + proofs**

◊ Dimensioning of Δ ⦊ Implementation of Δ + COTS

⦊ System S ⦊ Fielding of S

**Notation *M* stands for design computational model (in Π or in Δ)**

***M*: from pure synchronous to pure asynchronous**


FAIL
FAIL
FAIL

**Problem Π = {assumptions A (+ *M*); properties P; Cov(P)}**

**A = "adversary"of S** = run-time {system-centric, environmental} conditions. Typically, **A** =

- postulated models for {external event arrivals, internal processes, failures, …},
- postulated bounds on {external "loads", densities of failure occurrence, of intrusion occurrence, …}, wcet's (one per process), …

*M* is optional. May be chosen along with Δ.

**Cov(P)** = coverage of P under A (acceptable lower bound).

Coverage of α = likelihood or probability (α not violated at run-time).

For example, with safety-critical systems, $Cov(P) \approx 1 - 10^{-7}$.

# System abstraction levels (simplified view)

| | | |
|---|---|---|
| ENVIRONMENT | | |
| | | ← Assumptions A |
| APPLICATIONS | … | |
| **M/W** | **m** | ← **Properties P, Cov(P)** |
| OS/MONITORS | … | |
| …/… | k | |
| …/… | … | |
| END-TO-END COMs | c | |
| …/… | | |
| NETWORK AND I/O | 2 | |
| BASIC H/W (PHYSICS) | 1 | |

**Design solution Δ for Π =**
specification of {[modules] ∪ [architecture] ∪ [distributed algorithms]}

Modules, algorithms, perform **operations (comp/comm steps).**

Any Δ rests on **design assumptions (DA)**, such as, e.g., microprocessors, light speed, network reliability, …, + some **model** *M* (possibly stated in Π) that serves as a **design model**.

*M* encompasses abstraction levels from 1 (basic H/W) to m (M/W level).

Let C(α) stand for the highest achievable Cov(α).

**Two necessary conditions for proving that Π is correctly solved are:**

**C1: C(*M*) is computable, C2: C(*M*) > Cov(P).**

## MIDDLEWARE ISSUE EXPLORED:

**Should M/W aimed at systems/applications of interest be built out of partially/fully synchronous solutions or out of asynchronous solutions?**

**Which model *M* for designing M/W of interest?**

Computational model *M* ⇔ timing assumptions (TA(*M*))

**TA(*M*)** = **postulated upper** (lower) **bounds** for durations/delays of some **operations** (successful write in cache-memory, OS or M/W call/service, send message, propagate signal, sojourn in a waiting queue, …), at some **levels**, which **bounds are known at design time, and these bounds hold true from time 0.**

# 2. On computational models

***M* = Pure Asynchronous:**

No TA(*M*) at all postulated at any level ⇒no coverage issue involved.

Unfortunately, many problems in distributed fault-tolerant computing (P = SafeP + LiveP + DepP) have no deterministic solution in this model.

**⇒ "augmented" (pure asynchronous) models**

***M* = Asynchronous (Async):**

pure asynchrony "augmented" with some **time-free semantics**

⇒ M/W algorithmic solutions are time free

**$M$ = Partially Synchronous (ParSync):**

pure asynchrony "augmented" with TA($M$) stated at some levels – level m certainly in our case – for some operations.

These bounds appear in M/W solutions ⇒ some M/W algorithmic solutions are time free, others are time dependent

**Pure Synchronous (Sync) is a particular case of ParSync:**

Sync = TA($M$) stated for every level, for every operation. These bounds appear in M/W solutions (strictly time-dependent solutions).

Note: The TT approach [Kopetz 1998] is a particular case of Sync. Most delays are assumed to be constant rather than variable – which eliminates many difficult design issues (see section 3), prima facie. In fact, these issues must be addressed in order to prove that TT assumptions are correctly implemented, with coverage > Cov(P).

**ParSync:** e.g., the TA model [Cristian/Fetzer 1999], the TCB approach [Verissimo/Casimiro/Fetzer 2000]

TA and TCB rest on such variables as δ, which stands for an upper bound of **middleware level (m)** end-to-end interprocess message delays, and δ **is "guessed" or "estimated" or assumed.**

**Async:** models considered by many authors (including ourselves in [Hermant/Le Lann 2002]).

In particular, time-free semantics = Chandra/Toueg's "unreliable failure detectors" – FDs [Chandra/Toueg 1991]. Such semantics match **"low" levels, up to the end-to-end coms level (c) typically,** and these **semantics are assumed.**

Recall **C1: C(*M*) is computable,** **C2: C(*M*) > Cov(P).**

In order to implement any model *M* (with some computable coverage), one must prove some TimeP: postulated bounds TA(*M*) must match demonstrated bounds T(*M*) – with some computable coverage – at every level encompassed by *M*, to the exception of level 1.

**More precisely: One must solve a distributed real-time computing (DRT) problem – with some computable coverage – at every level encompassed by *M* (to the exception of level 1), even if initial Π is not a DRT problem – no TimeP in Π.**

**Bad news: Solving a DRT problem is notoriously difficult.**

## What is involved with solving a DRT problem $\Pi_k$ specified at some level k?

**Problem $\Pi_k$ = {assumptions $A_k$; $b_k$ bounds $TA_k(M)$; Cov($TA_k(M)$)}**

**$A_k$ = "adversary"of level k (of S)** = run-time {system-centric, environmental}conditions, derived from upper level (k+1):

- postulated models for {external and internal event arrivals, internal processes, failures, …} at level k,
- postulated bounds on {externally and internally generated "loads", densities of failure occurrence, of intrusion occurrence, …}, wcet's (one per process), …, at level k.

**Design solution $\Delta_k$ necessarily includes scheduling algorithm(s) Sched** (e.g. HPF, EDF, SSF, D-Over, …), for waiting queues inevitably build up – temporarily – in (distributed) systems, at every level.

## Proofs of (bounds) $TA_k(M)$?

⇒ **Conduct schedulability analyses**

- Identify worst-case scenarios that can be deployed by $A_k$ in the presence of $\Delta_k$
- Then, establish a computable analytical expression for each of the $b_k$ bounds $\mathbf{T_k}$ (a function of variables appearing in $A_k$, in particular) that are achievable in the presence of worst-case scenarios.

⇒ **Establish feasibility conditions (FC)**

$$\forall j, j \in [1, b_k]: \mathbf{Ta_{k,j}}(M) < \mathbf{T_{k,j}}$$

**Proofs and/or $\Delta_k$ rest on assumptions**

⇒ specifications of **adversary $A_{k-1}$**, of required timeliness properties $TA_{k-1}(M)$, i.e. specification of DRT problem $\Pi_{k-1}$.

Schedulability analyses are quite involved (highly combinatorial),

**especially with distributed systems**

[may explain scarcity of (constructive) publications/results]

For schedulability analyses, see scheduling theory, queuing theory, (max, +) algebra, constraint programming, combinatorial optimization, analytical calculus,…

Schedulability analyses rest on simplified models of {systems, software processes, COTS, …}, on approximated {processor architectures, internal adversaries, worst-case execution times, contention and/or queuing phenomena, …}, on "luxurious" assumptions sometimes

**⇒ computing the coverage of a schedulability analysis may turn problematic**

**Ideal:**

- necessary and sufficient FC (N&S FC)
- derived under perfectly accurate assumptions A and DA

$\Rightarrow$ $\theta_{k,j}$ **= tightest values of (correct) time bounds at level k (e.g., smallest achievable termination deadlines)**

**Real:**

simplified models are considered, postulated timings may not be "elementary", finding N&S FC is NP-hard, hence "simplified" (sufficient) FC

$\Rightarrow$ **$T_{k,j}$ = some computed bounds for level k**

**Two possible outcomes:**

- **Optimistic (i.e. incorrect) $FC_k \Rightarrow T_{k,j} < \theta_{k,j}$**

  **$\Rightarrow$ ParSync solutions are incorrect**

  (SafeP, LiveP are lost, in addition to losing TimeP)

Example with consensus: all messages sent in a round are received in $T_x$ at most; given that messages may take up to $\theta_x$, it is impossible to prove consensus (processes may decide differently, always)

- **Pessimistic (i.e. sufficient) $FC_k \Rightarrow T_{k,j} > \theta_{k,j}$**

  $\Rightarrow$ **ParSync solutions are correct, albeit "slow"** compared with (ideally achievable) executions that would be based on $\theta_{k,j}$ .

# Consequences

**C3: The smaller the number of levels encompassed by *M*, the lower the levels, the most likely it is that C(*M*) can be computed.**

**C4: The smaller the number of levels encompassed by *M*, the lower the levels, the easier it is to establish N&S FC.**

From C4, it follows:

**C5: The smaller the number of levels encompassed by *M*, the lower the levels:**

- **the higher the probability that FC are correct (i.e. Cov(*M*) ≠ 0),**
- **the smaller distance $T_k - \theta_k$, whenever FC are correct.**

## Two major differences between ParSync and Async

(1) $c < m$

(2) The DRT problems involved with ParSync fully depend on $\Pi$

$\Rightarrow$ they are not "adversary immune".

Conversely, the DRT problems involved with Async do not depend much on $\Pi \Rightarrow$ they are "adversary immune".

## Major difference between TA and TCB

In TA papers, there is no mention that schedulability analyses are mandatory in order to express and prove $\delta$

$\Rightarrow$ Cov(TA) cannot be computed

$\Rightarrow$ From C1 and C2, it follows that with TA, it is impossible to prove that initial problem $\Pi$ is solved.

Assertions such as "*a system alternates between "good" and "bad" periods*" are just wishes. **It may well be that a system designed in the TA model ends up being always incorrect or mute.**

Counter-argument: pick up some "really big" δ (e.g., 1 hour), and that δ will never be violated!

Wrong!! In the absence of FC, you cannot predict whether system S will or will not be "overloaded" or entering thrashing "too often". Whenever the case, real δ is infinite!!!

## Major difference n° 1: levels where DRT problems arise

| | | | |
|---|---|---|---|
| APPLICATIONS | … | | |
| M/W | m | | |
| OS/MONITORS | … | from 2 to m | |
| …/… | k | | |
| …/… | … | with | |
| END-TO-END COMs | c | | |
| …/… | | ParSync | from 2 to c with Async |
| NETWORK AND I/O | 2 | | |
| BASIC H/W (PHYSICS) | 1 | | |

# Major difference n° 2: Immunity to adversary

With ParSync:

- One must start from level m. Indeed, $A_m$ – for DRT problem $\Pi_m$ – is embodied in specification $\Pi$, which is defined at level m,
- Bounds $TA_m(M)$ are $TimeP_m$, i.e. those timeliness properties specified in $\Pi$.
- Bounds $TA_k(M)$, $k < m$, transitively depend on $TimeP_m$.

Note: Try to predict TimeP "guaranteed" by OS-level COTS products! Good luck ☺

With Async:

- One must start from level c. Indeed, $A_c$ – for DRT problem $\Pi_c$ – only partially depends on $A_m$.
- Bounds $TA_c(M)$ do not depend on $TimeP_m$.

This is so for the reason that the set of processes (σ) involved with some time-free semantics is not in the set of processes specified in initial Π. IOW, a designer is free to define set σ, as well as to pick up any scheduling scheme and/or rules for processes in σ.

Luck: Most accurate results regarding FC, with good coverage? Network-level communication problems (see, e.g., IEEE/ACM Trans. on Networking). In [Hermant/Le Lann 2002], one shows how to build Fast FDs – strong or perfect semantics – provably correctly (schedulability analysis given), out of network level timing assumptions. Other authors have established similar results for conventional FDs.

**Consequences**: **The DRT problems to be solved in order to implement Async provably correctly are (significantly) simpler than those "embodied" in Π, which are those to be solved with ParSync.**

## CONCLUSION:

## Async DOMINATES ParSync

**C3 ⇒Regarding computability of C(*M*), Async dominates ParSync**

**C5 ⇒Regarding (1) achievable values of C(*M*) or (2) tightness of demonstrated timeliness bounds T(*M*) – i.e. system "performance" – for some given C(*M*), Async dominates ParSync**

Nancy Lynch, 1996: "It is impossible or inefficient to implement the synchronous model in many types of distributed systems."

**⇒ "Synchronous Distributed" may be an oxymoron!**

Therefore, whenever problem Π is not a real-time computing problem,

**don't kill yourself**:

**consider or design asynchronous algorithmic solutions**

**⇒ coverage of SafeP, LiveP, DepP = Cov(Async)**

**⇒ for some problems, coverage issues (for SafeP, LiveP, DepP) do not arise at all with asynchronous solutions – see indulgent algorithms.**

**What if Π is a real-time computing problem?**

# 3. Asynchronous real-time: the “late binding” principle

## Asynchronous Real-Time?

## Timeliness properties achieved (and proven) with some solution designed in an asynchronous model.

Obvious contradiction? TimeP resulting from time-free solutions?

Inevitably, some time between when some problem Π is specified and when an implemented system-solution S is fielded, there must be a phase during which schedulability analyses must be conducted, which implies considering a Sync or a ParSync model.

Classical mistake: taking this to mean that *M* – the design model – has to be Sync or ParSync ⇒ "**early binding**" of Δ to Sync or ParSync.

In fact, design model *M* may be Async and implementation model (that matches S) may be Sync or ParSync ⇒ "**late binding**" of Δ to Sync or ParSync. It is (obviously) possible to characterize the worst-case temporal behavior of an asynchronous algorithm.

Note: TimeP proofs can be established only after SafeP and LiveP have been proven (no deadlocks, eventual termination, …, in addition to postulating wcet's for processes).

In case of "**early binding**", you are doing the opposite: prove TimeP first, and then use time bounds to prove SafeP and LiveP! With all the drawbacks presented before…

**Why is it that TT/Sync designs seem "simpler" and/or "safer"?**

"While ET systems are flexible, TT systems are temporally predictable", pp. 2-5, TTP/A-Protocol V2.00, Sept. 2002.

The reader is (obviously) invited to conclude that ET systems are not temporally predictable!!! IOW, scheduling theory, queuing theory, … are useless!!!

And we know that this cannot be right …

## The single-lane highway theorem

(sub-title: with TT solutions, no waiting queues)

Consider a highway with 3 lanes, reducing to 2 lanes, and then to 1 lane.

At rush hours, i.e. when highway resource allocation matters, car speed is highest on the 1-lane portion. Therefore, **highways should be 1-lane only!**

Translation: Waiting queues don't build up within TT systems, they build up outside TT systems (at entry points). So what? What matters to a user is knowing a bound on sojourn times in waiting queues, no matter where such queues happen to build up ☺
And due to lack of schedulability analyses with the TT approach, there is no way of predicting such bounds ☹

## The essence of TT vs. ET solutions

**For any given level k, ET solutions are faster and/or more efficient than TT solutions**

Example: a shared multi-access communication medium

- 10 message sources,
- each source generates 9 messages of duration 1 each, 1 message of duration 10, every 10 messages,
- up to x messages submitted concurrently, $1 \leq x \leq 10$

**TTP multi-access scheme = conventional Static_Sync_TDMA**

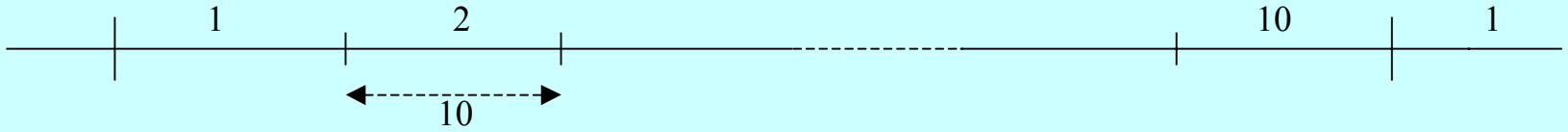
⇒ a pre-computed cyclic frame of 10 consecutive time slots,
each of duration 10

w = protocol overhead per frame + 10 inter-slot "guard times"

<u>Performance figures</u>:

T = worst-case waiting time(message) = 100 + w

Efficiency ratio (w ignored) = 0.19



## Deterministic Ethernets = conventional Async_TDMA

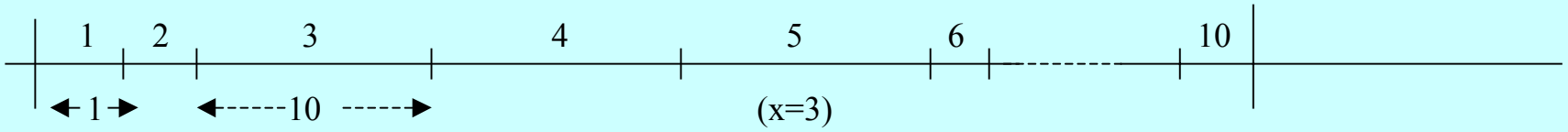
⇒ CSMA + deterministic contention resolution

w = protocol overhead per 10 messages

Performance figures:

T = worst-case waiting time(message) = 10 + 9x + w

Efficiency ratio (w ignored) = 1



This simple example is an illustration of well-known results in **computer networking**.

For problems in **distributed computing**, similar conclusions hold.

Reason? TT/Sync performance figures are max{max} functions, whereas ET/Async performance figures are min{max} functions.

# 4. Conclusions

**Asynchronous semantics are very much appropriate for capturing and solving real-world problems, including real-time problems, as well as for achieving the highest confidence figures.**

Very conservative users have seen the pitfalls of ParSync/Sync approaches. Some – e.g., French and European Space Agencies – even spend money on Async approaches.

FastUCS and FastUCN **Async algorithms** ([Hermant/Le Lann 2002]) are under development by a software company, soon to be delivered to ASTRIUM as **M/W components for new spaceborne applications**.

Demonstrator installed at ESA/ESTEC premises in June 2003.

Components to be made available to European industry by ESA/ESTEC.