Uncertainty and Predictability: can they be reconciled?

A Vision and a Concrete Model

Paulo Veríssimo
Univ. of Lisboa Faculty of Sciences
Lisboa – Portugal

pjv@di.fc.ul.pt
http://www.navigators.di.fc.ul.pt

The Vision

Problem Motivation

- Design and deployment of distributed applications is faced with the confluence of antagonistic aims:
  - between what is required by applications, and what is given by the supporting infrastructure/ environment
- Current and future large, massive-scale pervasive and/or ubiquitous computing systems will amplify this:
  - very high numbers of players, very large distances, geographical scope, topology and interconnections no longer a given, ill-defined COTS component properties
- Key lies with a changing notion of service guarantees:
  - about what have always been the fundamental issues, e.g., consistency, synchronism, reliability, availability, predictability, security, ...

Problem Motivation

- Take the time dimension
- Many services, beyond mere performance, have to secure *timeliness* properties, that is, they have to meet timing constraints (every x ms, within T, until T, etc.)
  - Dependability constraints: control applications; User-dictated QoS: Multimedia apps, synchronized groupware
- So we should use synchronous system models...:
  - But with unpredictable or unreliable infrastructures the system may fail
  - Dedicated infrastructures may be impractical or expensive
- Why not use asynchronous system models then?
  - They do not allow timeliness specifications

Grand challenges put by this scenario?

- Looks like a grand challenge would be withstanding uncertainty whilst achieving predictability
- Uncertainty:
  - ☞ is a common denominator of current systems
  - ☞ uncertain synchrony, fault model, and even topology
- Predictability:
  - ☞ systems are required to fulfil more and more demanding goals which imply predictability or determinism, e.g, timeliness, security
- Reconciling them means:
  - ☞ strong attributes (e.g. on ordering, agreement, timely termination of algorithms) can be secured in settings where usually very little is assumed and very little is expected from

Meeting the challenge

- Are there scientific advances we can look forward to?

a conflict must be solved between the weakness of the environment and the relative strength of requirements

- We propose to address this conflict with a "first-things-first" approach:

define adequate system model and architecture
before thinking about algorithms/protocols/mechanisms

Partial Solutions (in the time dimension)

- Models of intermediate synchrony have been around:
  - Partially synchronous (Dolev, Dwork)
  - Asynchronous with Failure Detectors (Chandra/Toueg)
  - Timed Asynchronous Model (Cristian/Fetzer)
  - Quasi-Synchronous Model (Veríssimo/Almeida)
- Did they solve the problem? Only partially ☺

A common feature we observed in these works: synchronism (or asynchronism) are not homogeneous properties of systems--- they vary with time and with space, i.e. the part of the system being considered

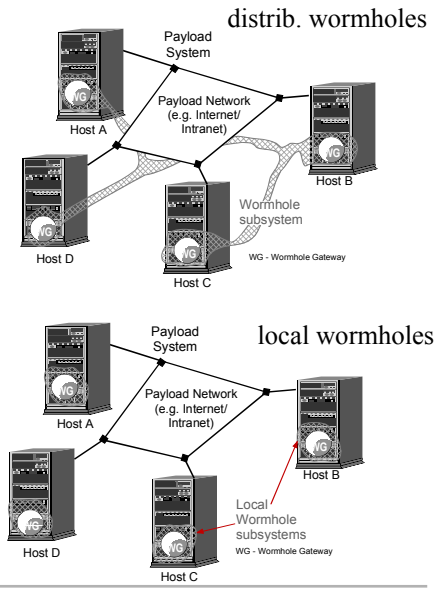
- *But how to control this process to our benefit?*

Guidelines

- Assume that *uncertainty is not ubiquitous* (and is not everlasting)--- the system has parts more predictable than others (and tends assume stable periods)
- Be *proactive in achieving predictability*--- make it happen at the right time, right place
- *Tolerate uncertainty* further to tolerating faults--- not all failures can be prevented, and some only on a probabilistic basis

Wormholes

- New design philosophy for distributed systems:
- constructs with privileged properties which endow systems with the capability of evading the uncertainty of the environment (``taking a shortcut'') for certain crucial steps of their operation, in order to achieve the required "hard properties" (predictability)



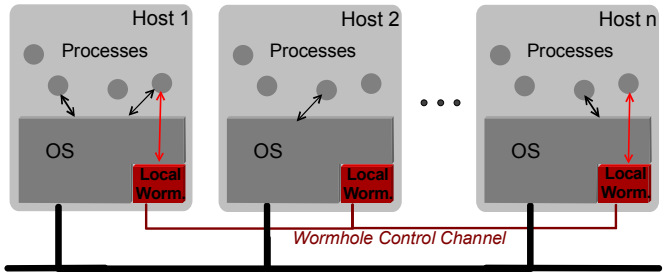
Characterisation of a Wormhole

- The little part that offers 'hard' properties, e.g.:
  - synchronous: bounds on processing delays, drift rate of local clocks and delivery delay of control messages
  - secure: trusted to be tamperproof, secure processing and comms.
- Small, simple and uses few resources
  - Easier to construct and verify, with high coverage
  - Supplies simple services, like failure detection, timely execution, trusted channels, or signatures
- Acts as a coverage amplifier for the whole system

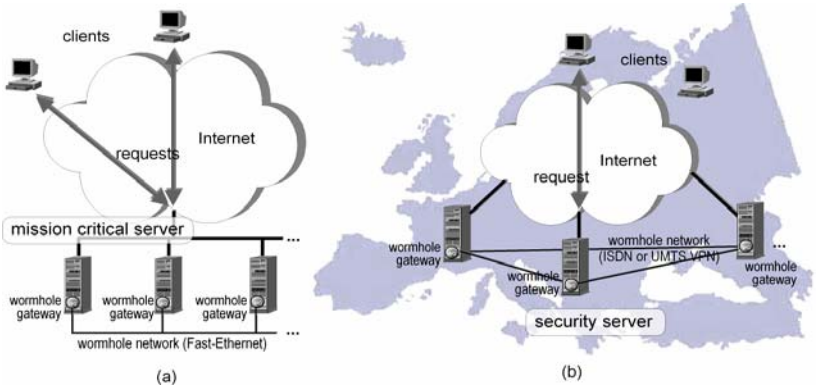
A small part of the system executes a small but critical part of its operation (a number of critical tasks) with high confidence (coverage)

Concrete examples

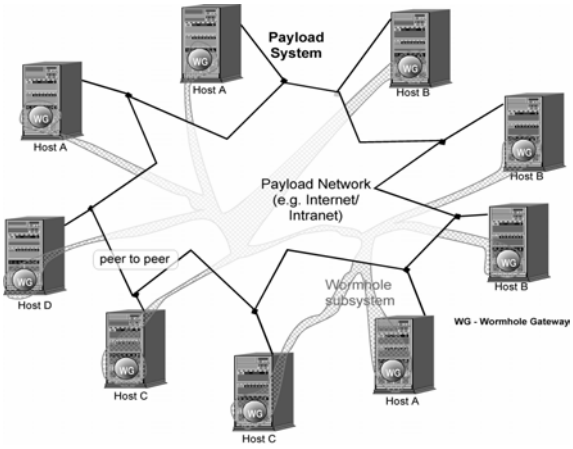
- We have played recently with two types of wormhole subsystems, to prove the concept:
  - Timely Computing Base for timeliness
  - Trusted Timely Computing Base for timeliness and security



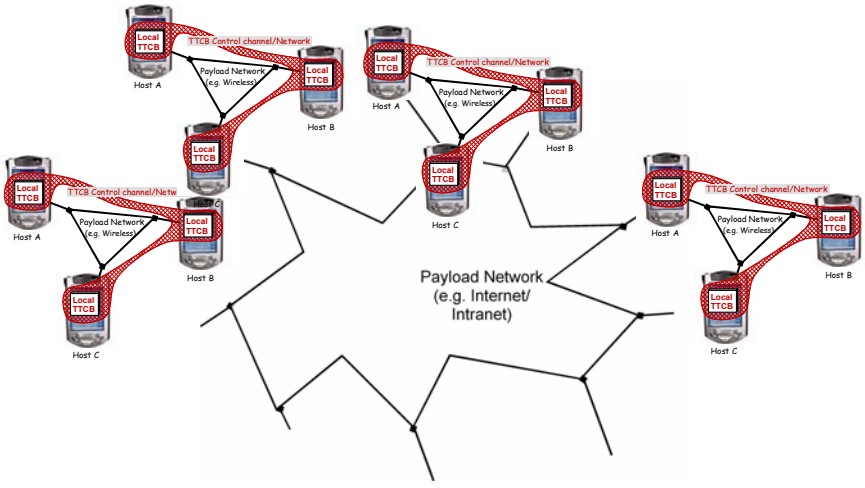
Example of deployment of systems with wormholes



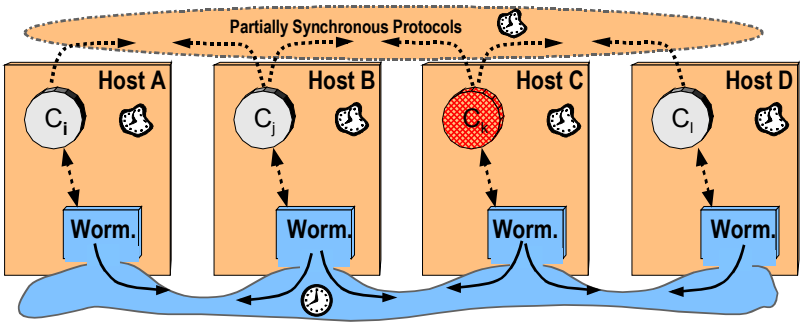
Example of deployment of systems with wormholes



Example of deployment of systems with wormholes

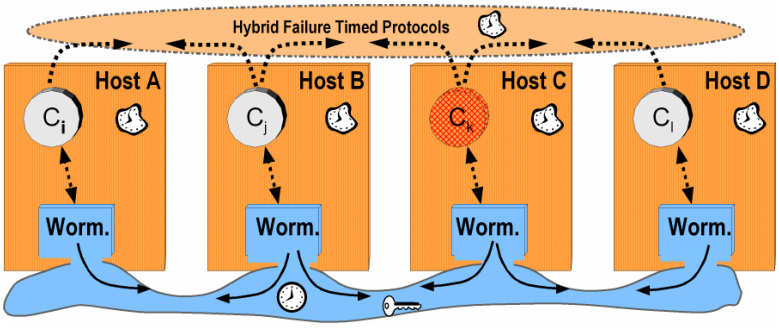


Strategy for timeliness awareness and/or assurance



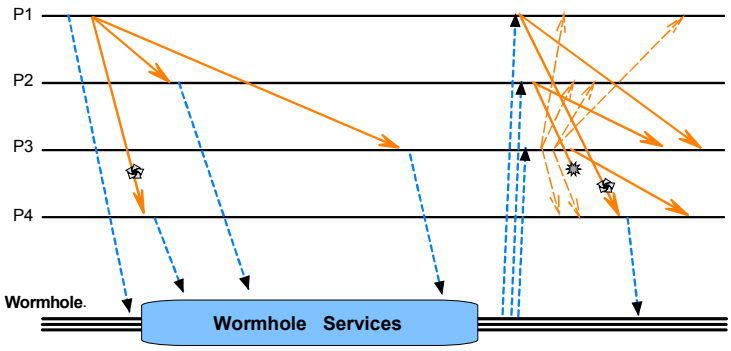
- - Fully synchronous, timely
- - Partially synchronous, potentially untimely

Strategy for security awareness and/or assurance



- - Fully secure (tamperproof), timely
- - Malicious, potentially untimely

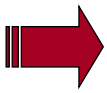
Rationale of the operation of Wormhole-aware protocols



A Concrete Model

(the timeliness wormhole example)

Timely Computing Base



TCB Model

TCB Services and Interface

Implementation of a TCB

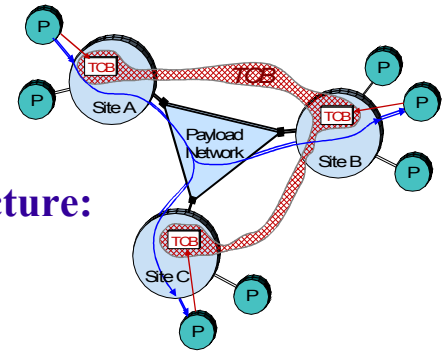
Quest for a Generic Solution

- How to encompass the entire spectrum of *synchrony,* from fully sync to fully async?
- How to enforce certain time-related properties rather than waiting for them to happen?

- We devised a model that encompasses the entire spectrum of *partial synchrony*
  - ☞ Timely Computing Base (TCB) Model
- We devised an architecture that enforces timeliness
  - ☞ Timely Computing Base (TCB) Architecture

The Timely Computing Base

- **Payload part:**
  - Where applications reside
  - Can have any degree of synchronism
- **Dual system architecture:**
  - Generic payload system
  - Control part: the TCB
- **Control part (TCB):**
  - Simple and small part of the system
  - Has known bounds on processing and message delivery delays



Why a new model

- None of the partial synchrony models is generic enough
  - each model treats synchrony asymmetries in it own way, relying on the evolution of synchrony with time, or with space, or both
  - neither can handle the whole synchrony spectrum
  - they tie application styles & semantics in one way or another

Why a new model (cont.)

- Timely Computing Base wormhole model :
  - ☞ postulates space-domain heterogeneity (vis-a-vis synchronism)
    - – ∃ components with "better" properties
  - ☞ monitors time-domain heterogeneity (synchrony variations)
    - – payload can have any synchronism
- Timely Computing Base architecture :
  - ☞ enforces architectural hybridization
    - – construction of the "better" components
  - ☞ enforces sync/async interface
    - – payload enjoying "better" components services

The conceptual "SYNC" toolbox

- TE – timely execution (simple or generic)
- DM – duration measurement (local or distr)
- CFD – crash failure detection (local or distr)
- TFD – timing failure detection (local or distr)

Alternative models

- Interesting applications for unrestrained failure modes require a perfect failure detector [Guerraoui02]
- A fully synchronous subsystem is required to implement a pFD [Chandra96]
- Weakest of known timed system is TA, but no pFD possible on strict TA
- Consider TA+SYNC, with simplest set of services: simpleTE, local DM
- Consider applications where slow nodes do *harakiri:*
  - Tout => Crash Failure (CF)
  - Tout => Local DM
  - *enforce*(Crash) => simpleTE
  - This version of TA+SYNC(simpleTE,locDM) can be implemented with a watchdog (WD)
  - pFD is achieved, through the transformation Tout => enforce(Crash), and some algorithmics to ensure that any P crashes before being suspected [Fetzer01]
- Consider applications that survive timing faults in one process and/or fail-safe requires orderly shutdown routines, and not just WD 'click'
- Then: WD is not generic TE, and Tout => *enforce*(Crash) is not acceptable

Alternative models

- Consider TA+SYNC(genTE, distrDM)
  - That is, we implement distr DM inside SYNC, gaining in precision and separation of concerns; we implemente generic timely execution
  - At this point, we can build pCFD inside the SYNC box
- Now we have TA+SYNC(genTE,distrDM, pCFD):
  - we can run all interesting non-timed applications
  - but can we build timely applications, even soft? No.
- Consider RT+SYNC(genTE, distr DM, distr pCFD):
  - Can we build timely appl's? Not generically, we need timing failure detection [Veríssimo99]
- Consider AnySyn+SYNC(genTE, distr DM, distr pTFD):
  - We can build any applic. from time-free to RT payload with the same model
  - A world of timed/timely applications opens
- Fully-fledged SYNC toolbox = TCB : gen TE, distr DM, distr TFD

Architecture and coverage

- How to build a SYNC toolbox, aka Timely Computing Base?
- Suppose environment with 't' parameters, and protocol 'P' which uses 't' directly, deriving property-level Tp [TTP, Cristian]
- If 't' is violated, P fails, sometimes not only timeliness properties, but also safety properties
- Suppose now environment with 't' parameters, and protocol 'P' which is implicitly indexed to 't'
- That is, P is time-free by construction, and there is a time complexity equation indexed to 't'
- P is *immersed* in the environment, and from there, actual values are derived for Tp [LeLann,AMp]

Achieving Synchronism by immersion

- How to achieve synchronism with a timer-driven ("async") protocol, given that classical approaches to synchronous protocols are clock-driven ("sync") [Cristian]?:
  - achieve and determine upper bounds on frame delivery delays by the abstract network, in the presence of ~~[illegible]~~
  - impose a performance specification on the NAC hardware/software (CPU, kernel, etc.) in order that processing times of the protocol actions are bounded, and known for the worst case traffic pattern specified;
  - structure the protocol in phases, so that an execution predictably has a bounded number of phases, clearly delimit phases, in what concerns error detection/recovery (omission and timing), and permanently ~~[illegible]~~ bounds);
  - structure each phase as a series of timed-out transmissions-with-response, so that it can be decomposed in time, in ~~[illegible]~~ deliveries and protocol actions as specified above, having thus with a known duration bound.

SYNCHRONISM OF ENVIRONMENT
TIME-FREE PROTOCOL STRUCTURE
IMMERSION

- With these measures, AMp execution time would be bounded to a known value [VeríssimoSRDS90]

Architecture and coverage

- Immersion is a good conceptual move, but ...
- Observe what happens when ‘t’ varies:
  - ☞ Protocols which have no timeliness properties will move faster or slower, depending on ‘t’; they will always be safe.
  - ☞ Protocols with timeliness (real-time) properties indexed by immersion to a given magnitude of ‘t’, will give timing failures when ‘t’ increases, violating timeliness.
  - ☞ Protocols whose safety properties depend on the environment being synchronous (‘t’ being respected) will fail on ‘t’ changing.

Architecture and coverage

- In the past discussion, there were two crucial protocols whose safety requires a fully sync environment (‘t’ not failing): pCFD and pTFD
- Any appl. running these protocols on the same environment used by normal (payload) protocols, has a coverage problem for a start:
  - ☞ The whole system is complex, so coverage comes down
  - ☞ How do we enforce ‘t’ for complex systems? ☹

Architecture and coverage

- Suppose the system is hybrid with regard to fault and synchrony
  - Payload: any synchrony, indulgent w.r.t. 't_P' of the environment
  - Control: fully synchronous, strict w.r.t. 't_C' of the environment
  - It is built through architectural hybridization:
    - the part of the environment supplying 't_C' is specially built: we do get 't_C'!
    - the rest (payload) is normal stuff: 't_P' is not so assured
- Build control algorithms (ex., pCFD, pTFD) on control part
- Immerse payload algorithms (e.g. time-free) into the 't_P' environment.
- 't_P' hypothesis may fail, but can be detected by control (ex., pCFD, pTFD) with high assurance (of 't_C')

Dependability framework for adaptivity

Dependability framework for adaptivity

- Restate 'correctness' definition:
  - Consider normal and critical properties
  - Normal properties can be violated within rules
  - Critical properties cannot
- Define behaviour classes:
- Adaptive
  - Recurrent violation of a normal property is accepted, with a bounded probability
- Safe
  - Occasional violation of a normal property is accepted, its up to the system to react (e.g. using conventional fault tolerance)
- Fail-safe
  - Any violation of a property is not acceptable, the system must stop

What is the philosophy?
(time domain example)

- Timing failures more complex than they look
  - **Unexpected delay** - "normal" effect
  - **Contamination** (of safety props) - error propagation effect
  - **Decreased coverage** - continued (statistical) effect
- Can we achieve correct operation despite these?
  - Contamination should be avoided at all cost
  - Coverage should remain stable

Why is the framework generic

- Generic properties dictate correctness of applications, regardless of functional semantics
  - **Coverage Stability -** coverage of timing assumptions remains stable
  - **No-Contamination -** safety properties not violated
- Under uncertain timeliness, different classes of appl's secure these properties in different ways
- It is necessary to detect timing failures, and react to that

TIMING FAILURE DETECTOR (TFD) considered fundamental

Dependable and Adaptive Computing with a TCB wormhole

- Introduce classes of applications that deal with these problems when assisted by a TCB:
  - Fail-safe: exhibits correct behaviour or stops in fail-safe state
  - Time-elastic: exhibits coverage stability
  - Time-safe: exhibits no-contamination
- Apply known fault tolerance techniques to the application classes (or combinations thereof):
  - detection and/or recovery; masking
- *The TCB Model and Architecture* [ieeeTOCS2002]

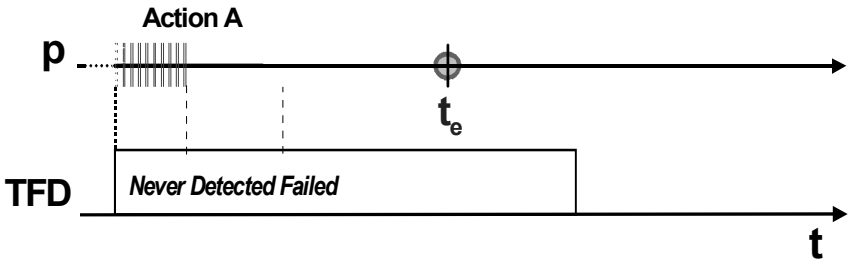
Example application frameworks

- Fail-safe operation [DSN2000] :
  - by switching to a fail-safe state after the first failure
  - requires the TFD service and appl´s to be of the fail-safe class
- Reconfiguration and adaptation [SRDS2001] :
  - by enforcing coverage stability
  - requires appl´s to be of the time-elastic and time-safe class
- Timing error masking [DSN2002]:
  - by using replication to mask transient timing errors
  - requires the TFD service and appl´s to be time-safe class

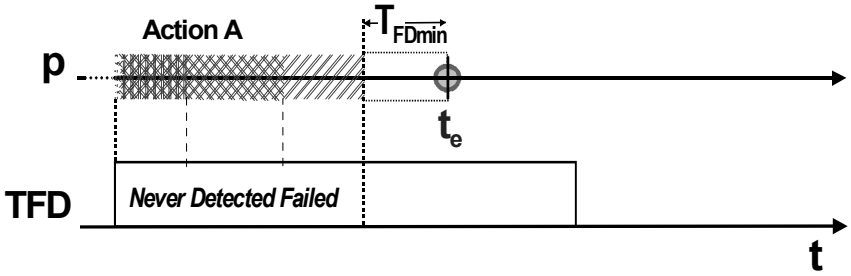
Timing Failure Detection

- Timing Failure:
  - ☞ Given the execution of a timed action X specified to terminate until real time instant te, there is a timing failure at p, iff the termination event takes place at an instant t'e, te < t'e ≤ ∞
- Timing Failure Detection:
  - ☞ **Timed Strong Completeness:** *There exists* TTFDmax *such that given a timing failure at* p *in any timed action, the TCB detects it within* TTFDmax *from* te
  - ☞ **Timed Strong Accuracy:** *There exists* TTFDmin *such that any timely timed action that does not terminate within -* TTFDmin *from* te *is considered timely by the TCB*

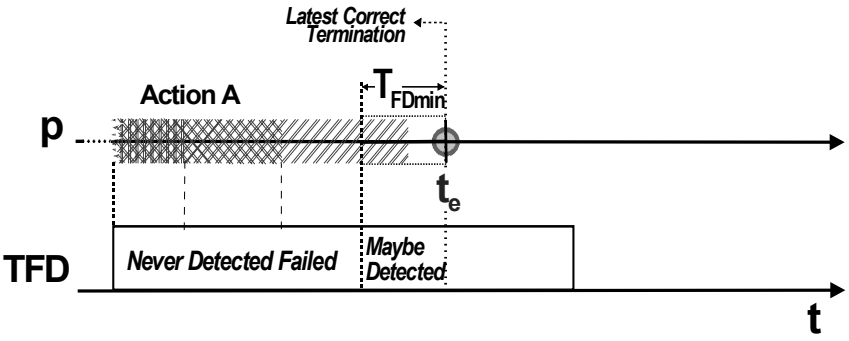
Timing Failure Detection



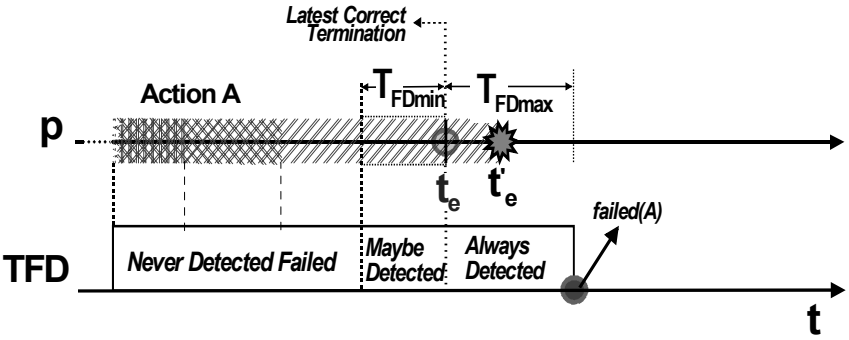
Timing Failure Detection



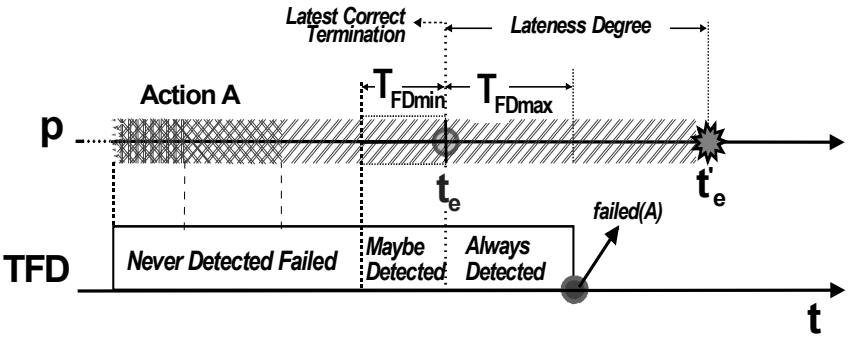
Timing Failure Detection



Timing Failure Detection



Timing Failure Detection



TCB Model

TCB Services and Interface

Implementation of a TCB

Services and API of the TCB

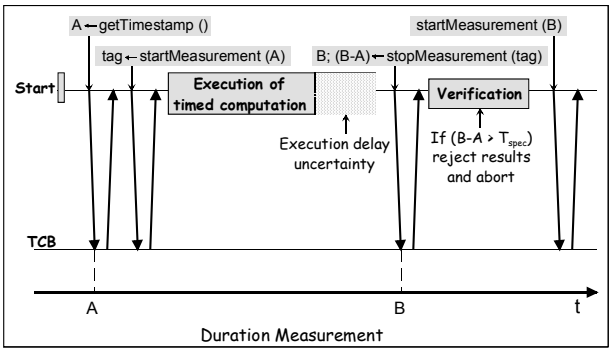
- The TCB provides minimal services:
  - TE - Timely execution
  - DM - Duration measurement
  - TFD - Timing failure detection
- And a payload-to-TCB interface
  - Allows potentially asynchronous applications to dialogue with a synchronous component

Synch/Asynch Interface

- Important issues to retain:
  - The TCB does not make applications timelier
  - Service invocation latency not bounded
  - Service responses or timing failure notifications not bounded
  - Nothing obliges applications to become aware of failures
- The TCB as an oracle:
  - Applications take advantage of the TCB by construction
  - They observe correctness of past steps before proceeding
  - Timeliness always observed in terms of durations
  - Time-critical responses to failures handled by the TCB

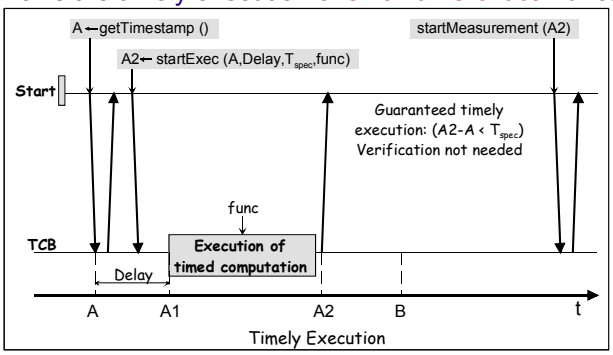
API

- Duration Measurement
  - ☞ Allows the measurement of upper bounds of payload actions



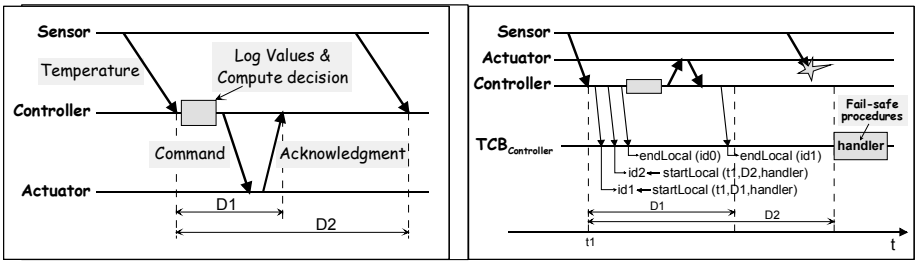
API

- Timely Execution
  - ☞ Allows the timely execution of small time-critical functions



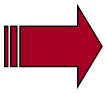
API

- Timing Failure Detection
  - Allows detection of timing failures (of local and distributed actions)
  - Allows timely execution of safety procedures upon the detection of a timing failure



TCB Model

TCB Services and Interface

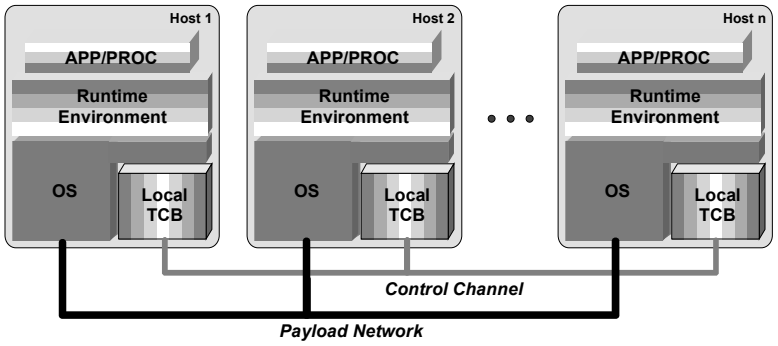


Implementation of a TCB

Implementation Issues

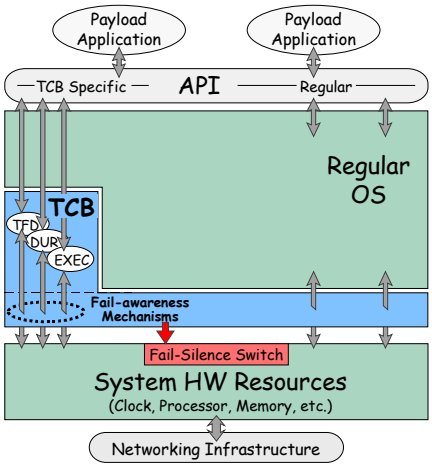
- Duration measurement
  - Local durations: local clock
  - Distributed durations: round-trip duration measurement technique
- Timely execution
  - Small functions residing in the TCB address space
  - Use known techniques of real-time systems
    - Admission control, schedulability analysis, WCET calculation
- Timing failure detection
  - Distributed protocol to ensure (Timed) Completeness and Accuracy
  - Set up timeouts using timers (based on local clock)

A concrete example



A System with a TCB

- The TCB is built with given <*bound,coverage*> pairs
- Hardware and inter-TCB communication channel may assume different forms, for different <*bound,coverage*> pairs
- Software kernel on a plain desktop (PC or workstation) may be used



Conclusions

- Some achievements...
- (T)TCB wormhole prototypes
  - Software Available at http://www.navigators.di.fc.ul.pt/software/tcb
- Timing fault tolerance for event-based systems
- Byzantine-resilient reliable multicast
- In preparation: Byzantine-resilient consensus, atomic multicast and membership
- See more at: www.navigators.di.fc.ul.pt -- “Documents”

Some Recent Publications

- See more at: www.navigators.di.fc.ul.pt -- "Documents"

- *Traveling through wormholes: Meeting the grand challenge of distributed systems*. Paulo Veríssimo. In Proceedings of the International Workshop on Future Directions in Distributed Computing, pages 144–151, June 2002
- *The Timely Computing Base Model and Architecture*. P. Veríssimo, A. Casimiro. Transactions on Computers - Special Issue on Asynchronous Real-Time Systems, vol. 51, n. 8, Aug 2002
- *The Design of a COTS Real-Time Distributed Security Kernel*. Miguel Correia, Paulo Veríssimo, Nuno Ferreira Neves. 4th EDCC, Toulouse, France, October 2002
- *Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model*. Miguel Correia, Lau Cheuk Lung, Nuno Ferreira Neves, Paulo Veríssimo. 21st Symposium on Reliable Distributed Systems, Suita, Japan, October 2002
- *Generic Timing Fault Tolerance using a Timely Computing Base*. António Casimiro, Paulo Veríssimo. Proceedings of the International Conference on Dependable Systems and Networks, Washington D.C., USA, June 2002
- *Using the Timely Computing Base for Dependable QoS Adaptation*. António Casimiro, Paulo Veríssimo. Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems, New Orleans, USA, October 2001
- *CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities*. Paulo Veríssimo, V. Cahill, António Casimiro, K. Cheverst, A. Friday, J. Kaiser. Proceedings of European Wireless 2002, Florence, Italy, February 2002