Open-Source in Dependable Systems: *a problem or a solution?*

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

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

"Open source" COTS?

- As far as open source vs. COTS are concerned, the questions become interestingly antagonistic
- Open-source (COTS) considered a solution:
- Open source may allow to know COTS better, and make better and more robust COTS
- Open-source (COTS) considered a problem:
- Open source deterrent of quality and permissive of tampering, leading to worse COTS
- Our viewpoint:
- with the adequate perspective, both open-source and black-box COTS can be combined harmoniously in robust and cost-effective designs. We discuss the *perspective* in the next slides

Putting the subject in context

- We will try to show four things:
- being “COTS” or “open-source”, orthogonal issues
  - We can have *open-source COTS* (e.g. Linux)
- COTS essential and a part of life
  - They’re the trees of our system forests
- open-source and black box COTS useful
  - We need both wild and green-house trees
- right architectural approach fundamental
  - look at the forest, don’t forget the trees, allow biodiversity
  - this going to be the main point made in this talk

What is the right hook?

- Distributed and modular systems changed the way we look at architectures:
  - made possible the advent of COTS
  - made us have to take COTS problems into account
- COTS components issues:
  - fashionable and almost mandatory in the design of current modular systems
  - sometimes questioned in a dependability context
  - e.g. for having ill-defined properties and behaviour

Meeting the challenges

- *"The system architect, today, assembles pieces of hardware that are at least as large as a computer or a network router or a LAN hub, and assigns pieces of software that are self-contained, such as client or server programs, Java applets or protocol modules, to those hardware components"*
  - ☞ [Distributed Systems for System Architects, Kluwer, 2001]
- *"…applications that operate independently of direct human control, and networked mobile intelligent components that may organise themselves into autonomous, mobile and rapidly compoundable co-operating communities, forming an unstable and mobile object population, unpredictable network load, varying connectivity, presence of failed system components."*
  - ☞ [CORTEX project vision, 2001]

- Solution to these problems must lie within modular system architecture principles:
  - ☞ a given composition of components, with given functional and non-functional properties and an interface where these properties manifest themselves.
  - ☞ components are placed in a given topology of the architecture, and interact through algorithms/protocols such that global system properties emerge from these interactions

In search of architectural thinking

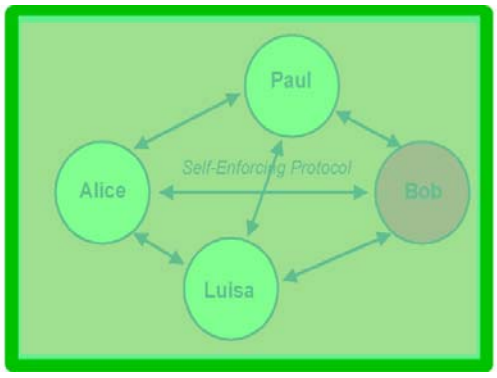
- "what are the appropriate levels to insert F/T to HW SEUs?" [J. Abraham]
- "redundancy by duplication is costly and may not work" [R. Iyer]
- "programmable embeddable appl-specific security/error checkers" wanted [R. Iyer]
- systems should be endowed with "compiler-based and hw supported programmable vulnerability masking/ error detection techniques" [R. Iyer]
- "control and overlay networks solve congestion and/or security problems" [Cisco, Akamai]

  - ☞ can we make strong statements about the effectiveness of these measures (e.g. prove properties) without architectural thinking?

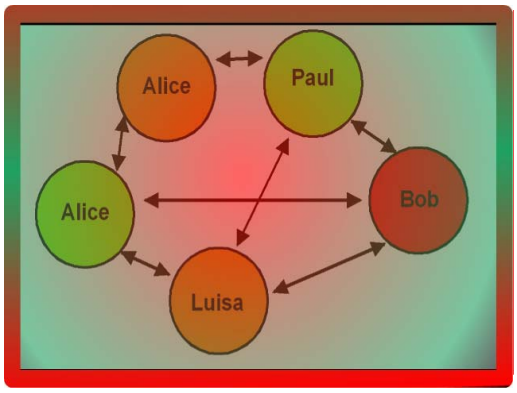
Dangers of ignoring architecture and modularity

- The “theoretical” approach-- no assumptions:
  - ☞ inefficient or not powerful design
- The “engineer” approach-- unjustified assumptions:
  - ☞ lack of coverage of design
- The “architect” approach-- homogenous assumptions:
  - ☞ the good attitude, but lack of versatility if just good old layering approach

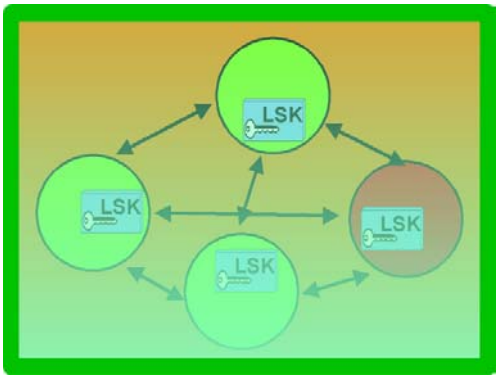
Of course, these issues are amplified in adverse environments such as in the presence of malicious faults



- Arbitrary model – no assumptions
- High coverage, low efficiency



- Fail-controlled model with non-justified environment assumptions
- Fair to low coverage – no enforcement



- Architectural-hybrid model – little environment assumptions; justified component assumptions
- High coverage – enforcement by a few “good” components

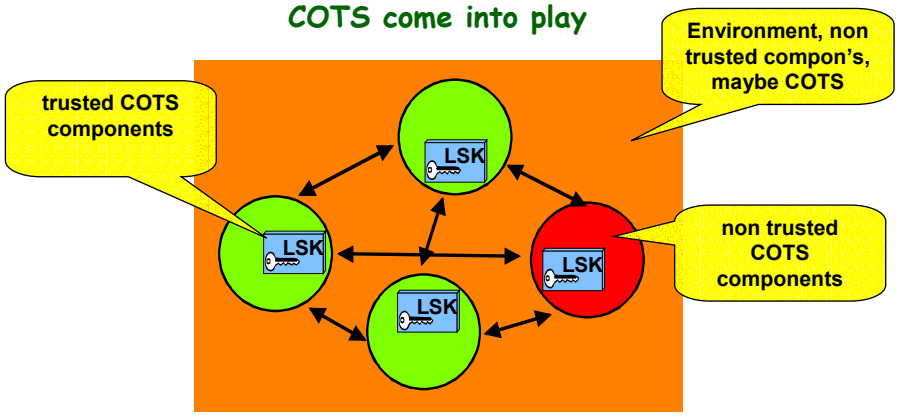
Architectural hybridization

- *presence and severity of faults varies from component to component and is effectively constrained by the component's trustworthiness (or dependability)*
- *failure assumptions are in fact enforced by the architecture and the construction of the system component, thus substantiated*

- Based on *modular view* of systems
- Based on a *dependability-awareness* perspective
- Postulates *recursive use of fault tolerance and fault prevention*
- Natural compatibility with *component-based design* towards dependable COTS-based systems

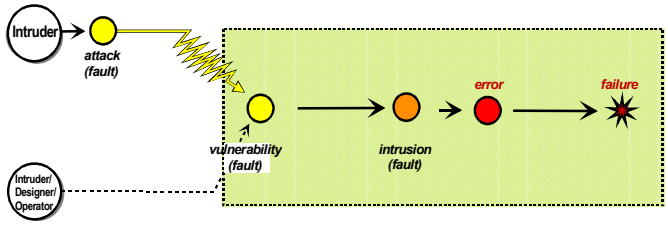
Intrusion-Tolerant Architectures: Concepts and Design. Paulo Veríssimo, Nuno Ferreira Neves, Miguel Correia. In: Architecting Dependable Systems. Springer-Verlag LNCS 2677 (2003)

COTS come into play



- Non trusted COTS
  - ☞ The black-box ones? certainly for those we don't know much about
- Trusted COTS
  - ☞ The open source ones? Certainly for those we must make trustworthy! Also helps in case we want to review them and make sure we can trust them

How do we improve COTS subsystems?



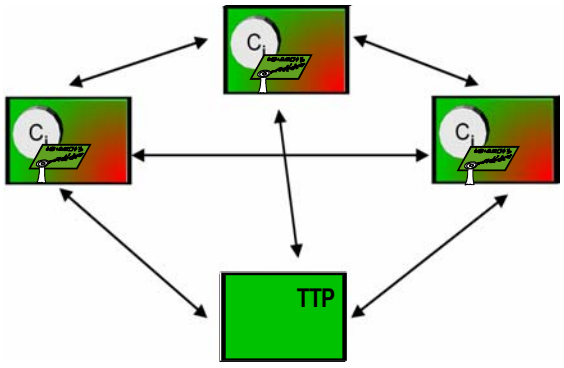
Let the fault model guide our architectural assumptions

Attack-Vulnerability-Intrusion composite fault model

AVI sequence : *attack + vulnerability→ intrusion → error → failure*

- Example:
  - prevent/tolerate attacks matching vulnerabilities you can't remove
  - remove/prevent vulnerabilities matching attacks you can't prevent

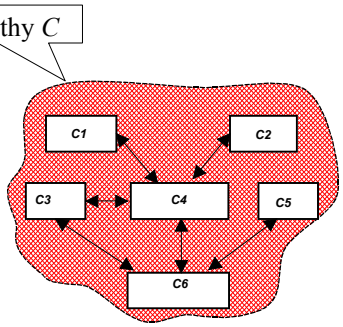
How do we improve COTS subsystems?



- Intrusion tolerant TTP server using COTS comp's:
  - Recursive use of F. Prevention and F.Tolerance
  - Work at subsystem level to achieve justifiable behaviour
  - Prevent certain client behaviour using certificate authentication
  - Build TTP server using replicated COTS and I/T techniques

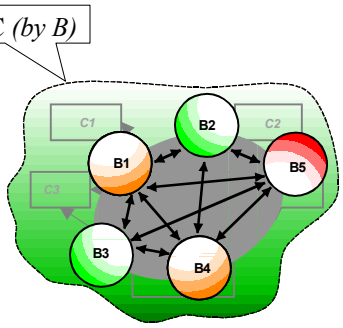
Dependability-aware design

- Architectural hybridization:
  - failure assumptions enforced by architecture and construction, thus substantiated
  - combined/recursive use of attack/vulnerability/intrusion prevention/removal/tolerance
- Trusted (trustworthy) components:
  - components or subsystems with justified coverage, used in the construction of fault-tolerant protocols under architectural hybrid failure assumptions

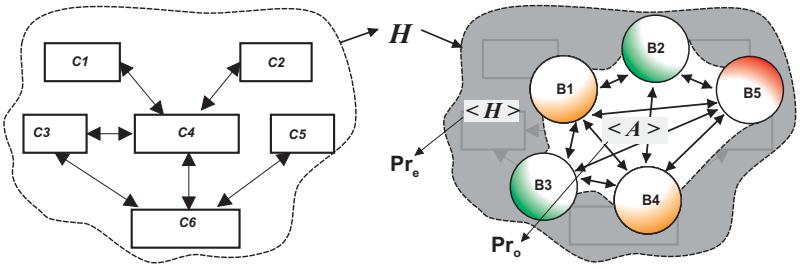


Dependability-aware design

- Architectural hybridization:
  - failure assumptions enforced by architecture and construction, thus substantiated
  - combined/recursive use of attack/vulnerability/intrusion prevention/removal/tolerance
- Trusted (trustworthy) components:
  - components or subsystems with justified coverage, used in the construction of fault-tolerant protocols under architectural hybrid failure assumptions



Building trust

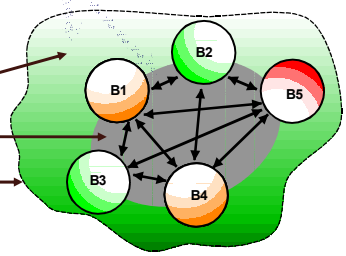


Intrusion-Tolerant Architectures: Concepts and Design. Paulo Veríssimo, Nuno Ferreira Neves, Miguel Correia. In: Architecting Dependable Systems. Springer-Verlag LNCS 2677 (2003)

On coverage and separation of concerns

- predicate P holds with a coverage Pr
  - we say that we are confident that P has a probability Pr of holding
- environmental assumption coverage (Pre)
  - set of assumptions (H) about the environment where system will run
  - *Pre = Pr (H | f)* *f- any fault*
- operational assumption coverage (Pro)
  - the assumptions about how the system/algorithm/mechanism proper (A) will run, under a given set of environmental assumptions
  - *Pro = Pr (A | H)*

Pr(A) = Pro x Pre = Pr (A | H) x Pr (H | f)

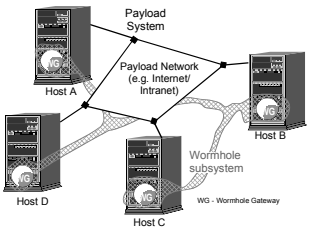


A few examples

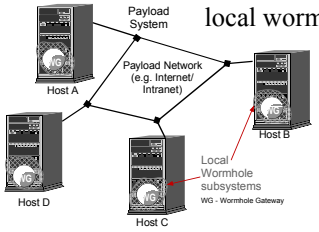
Wormholes

- 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)

distrib. wormholes



local wormholes

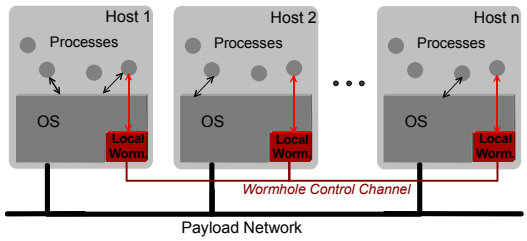


•*Uncertainty and Predictability: Can they be reconciled?* **Paulo Veríssimo**. Future Directions in Distributed Computing, Springer-Verlag LNCS 2584, 2003

Wormholes can be made of COTS

Wormholes seen from inside

- 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



Using Real-Time Linux to build a Timely Computing Base

i.e. a COTS wormhole specialized in being an assistant for reliably performing real-time (timely) embedded functions (interval measurement, failure/error detection, diagnosis, etc.)

Objectives

- Implement a TCB prototype using:
  - an infrastructure composed of normal Pentium PCs
  - the Real-Time Linux operating system
  - a switched Fast-Ethernet LAN
- Investigate:
  - the potential of RT-Linux to provide a real-time behavior
  - the requirements for real-time communication using a switched Fast-Ethernet network
- Obtain:
  - an intuition of what can be expected in terms of timeliness, based on experimental measurements

RT-Linux Overview

- Low level approach for Unix-like RT-OS
  - Real-time kernel underneath the operating system
  - Linux runs as another real-time task, and can be preempted
- Key mechanism: virtual interrupt scheme
  - Modified interrupt table vectors (now pointing to RT-Linux routines)
  - Modified `cli` and `sti` interrupt macros (now execute RTL code)
  - Linux can no more disable interrupts for an unpredictable time
- Only basic services provided:
  - low-level task creation, installation of ISRs, communication queues
  - I/O interactions require the design of specific RTL drivers

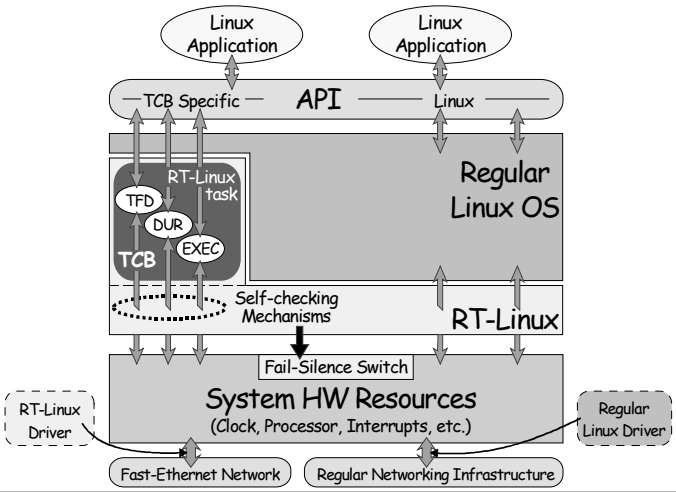
How Real-Time is RT-Linux?

- Potential predictability problems in RTL:
  - DMA data transfers: have priority over the CPU on the access to the system bus, introducing small but unpredictable execution delays
  - Interrupt requests: are handled by the RT-Linux virtual interrupt layer, but nevertheless introduce an unpredictable delay
- Possible solutions:
  - avoid DMA transfers (not always possible)
  - disable interrupts during critical executions (may have side-effects)
  - implement safety procedures (according to the validation principle the TCB is able to detect violated bounds and may apply safety measures)

RT-Linux Ethernet Driver

- Communication delays depend on driver
- RT-Linux network driver:
  - based on the standard Linux driver for 3COM cards
  - interface with the system designed for real-time operation
  - buffer management specifically designed for TCB needs
  - timestamp generation (send, receive) for self-checking purposes
- Driver services:
  - Transmission: only broadcast; replication to mask net omissions
  - Reception: messages stored in buffers; assumes periodic consumption to avoid buffer overflows
  - Transmission and consumption periods must be adapted

RT-Linux TCB

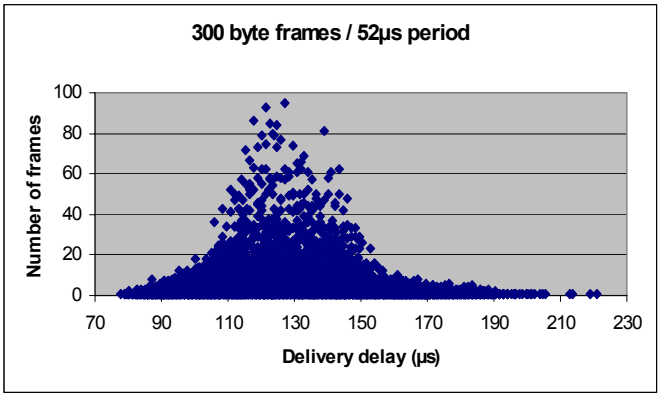


Experimental Results

	Transmission period				
	Zero-loss period				
Frame size	52µs	113µs	168µs	1ms	10ms
100 bytes				42-47	42-49
300 bytes	78-221			84-122	83-90
680 bytes		153-258		153-178	162-177
1024 bytes			221-348	221-359	231-279

Message delivery delays (µs)

Experimental Results

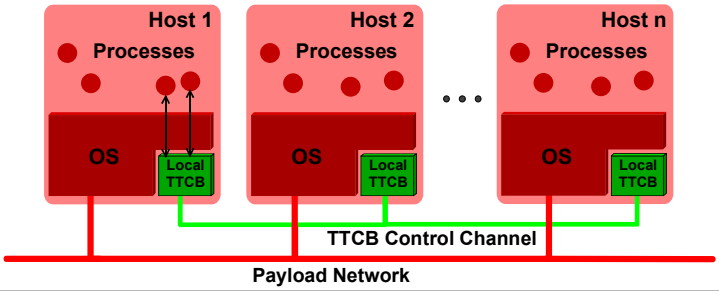


The Design of a COTS Real-Time Distributed Security Kernel

i.e., a COTS wormhole specialized in being an assistant for performing secure embedded functions (trusted binary block broadcast/consensus, trusted timestamping, failure/error detection, diagnosis, etc.)

Summary: purpose of the TTCB

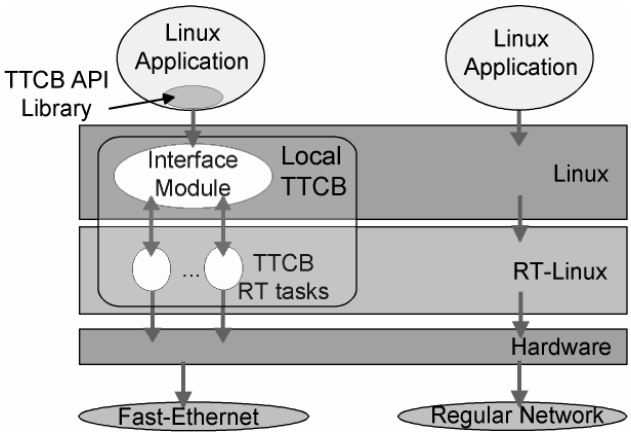
- Support impl. of intrusion tolerant protocols/apps
  - Run mostly in the payload system (can be attacked)
  - Use the TTCB to perform critical steps (secure)



Summary: COTS-based TTCB

- The paper presents the **TTCB model** and the design of **a TTCB**, the **COTS-based TTCB**:
  - Standard PCs
  - Real-time kernel: RT Linux / RTAI
  - Fast-Ethernet
- Basic idea of the design:
  - Local TTCB inside the kernel
  - Protect the kernel (remove vulnerabilities)
  - Protect access to the TTCB network
- Assumption coverage..., easy to deploy

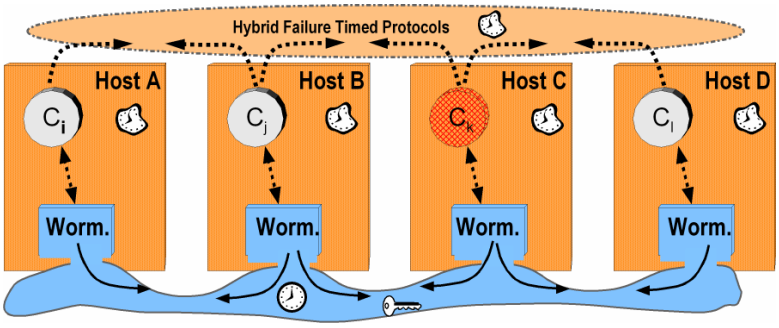
COTS-based Local TTCB Architect.



The fun only starts now

Looking at architecture again...

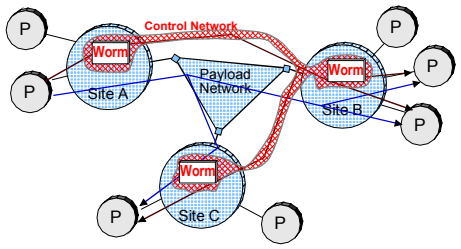
Strategy for security awareness and/or assurance



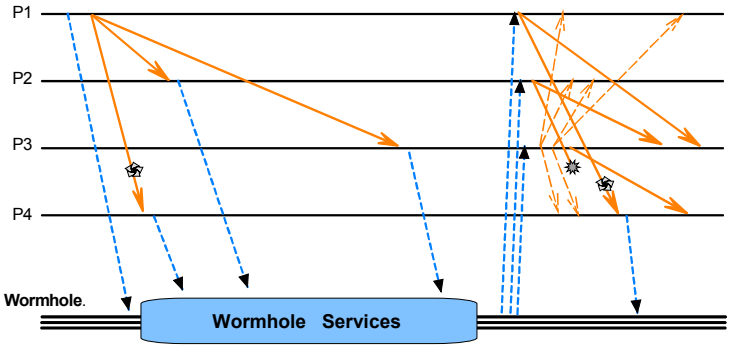
- Fully secure (tamperproof), timely
- Malicious, potentially untimely

Wormholes seen from inside

- 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



Rationale of the operation of Wormhole-aware protocols



Conclusions

- Without "open-source"
  - ☞ these implementations would not be possible
- Without COTS
  - ☞ these implementations would not be maintainable
- Ability to define a behavioural interface with precision even if with weak assumptions:
  - ☞ open-source- ☺ ; black-box- ☹
- Ability to improve on a component's deficiencies and make it more robust:
  - ☞ open-source- ☺ ; black-box- ☹
- (T)TCB wormhole prototypes
  - Software Available at http://www.navigators.di.fc.ul.pt/software/tcb
- 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

Some Recent Publications (urls)

- *Modeling Wormholes*
- *Uncertainty and Predictability: Can they be reconciled?* **Paulo Veríssimo**. Future Directions in Distributed Computing, Springer-Verlag LNCS 2584, 2003
- *The Timely Computing Base Model and Architecture*. **Paulo Veríssimo, António Casimiro.** IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems, vol. 51, n. 8, Aug 2002
- *The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness*. **Paulo Veríssimo, António Casimiro, C. Fetzer.** In Proceedings of the 1st International Conference on Dependable Systems and Networks, New York, USA, June 2000.
- The Timely Computing Base. Paulo Veríssimo and António Casimiro. **Technical Report DI/FCUL TR 99-2, Department of Informatics, University of Lisboa,** May 1999. *(original paper, improved in TOCS02)*

- *Implementing Wormholes*
- *Measuring Distributed Durations with Stable Errors*. **António Casimiro, Pedro Martins, Paulo Veríssimo, Luís Rodrigues.** Proceedings of the 22nd IEEE Real-Time Systs Symposium, London, UK, December 2001
- *How to Build a Timely Computing Base using Real-Time Linux*. **António Casimiro, Pedro Martins, Paulo Veríssimo.** in Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems, Porto, Portugal, September 2000.
- *Timing Failure Detection with a Timely Computing Base*. **António Casimiro, Paulo Veríssimo.** 3rd Europ. Research Seminar on Advances in Distr. Sys (ERSADS'99), Madeira Island, Portugal, April 23-28, 1999
- *The Design of a COTS Real-Time Distributed Security Kernel*, ***Miguel Correia, Paulo Veríssimo, Nuno Ferreira Neves**, Fourth European Dep. Comp. Conf., Toulouse, France, October 2002 © Springer-Verlag.*

- *Using Wormholes*
- *Using the Timely Computing Base for Dependable QoS Adaptation*. **António Casimiro, Paulo Veríssimo.** Proceedings of the 20th IEEE Symp. on Reliable Distributed Systems, New Orleans, USA, October 2001
- *Generic Timing Fault Tolerance using a Timely Computing Base*. **António Casimiro, Paulo Veríssimo.** Procs of the Intern'l Conference on Dependable Systems and Networks, Washington D.C., USA, June 2002
- *Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model*, ***Miguel Correia, Lau Cheuk Lung, Nuno Ferreira Neves, Paulo Veríssimo.** Proc's of the 21st Symp. on Reliable Distributed Systems (SRDS'2002), Suita, Japan, October 2002*