Dependability, Security, two faces of a same coin?

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

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

This is not a tutorial on intrusion tolerance

- But it does wander around the concept, for obvious reasons.
- However, a tutorial on intrusion tolerance, and a companion text, are available from the University of Lisboa Technical Reports web site.
- http://www.navigators.di.fc.ul.pt/it

Some philosophy for a start

- What characterizes a dependable system?
  - A set of safety and liveness properties
- What characterizes a secure system?
  - A set of safety and liveness properties
- What may impair a dependable system?
  - A set of faults -> failure
- What may impair a secure system?
  - A set of faults (attacks, vulnerabilities, intrusions) -> failure
- How do I make a system dependable (normally)?
  - Using fault avoidance (prevention, removal) and fault tolerance (error detection, recovery, masking)
- How do I make a system secure (normally)?
  - Using fault avoidance (attack prevention, vulnerability removal)
  - and some bits of fault tolerance (intrusion detection)
  - Nowadays, increasingly fault tolerance (intrusion detection, recovery, masking)

What would Intrusion Tolerance be?

- Traditionally, security has involved either:
  - Trusting that certain attacks will not occur
  - Removing vulnerabilities from initially fragile software
  - Preventing attacks from leading to intrusions
- In contrast, the tolerance paradigm in security:
  - Assumes that systems remain to a certain extent vulnerable
  - Assumes that attacks on components or sub-systems can happen and some will be successful
  - Ensures that the overall system nevertheless remains secure and operational
- In other words:
  - Faults-- malicious and other--- occur
  - They generate errors, i.e. component-level security compromises
  - Error processing mechanisms make sure that security failure is prevented

Some non-negligible difficult bits…

- **Hackers:**
  - they don't behave as we wish or predict…
  - ... God damn them!
- **Modeling the nature of faults**
- **Putting the coverage problem in perspective**
- **The substance of fault models**
- **Synchrony models in the presence of malice**

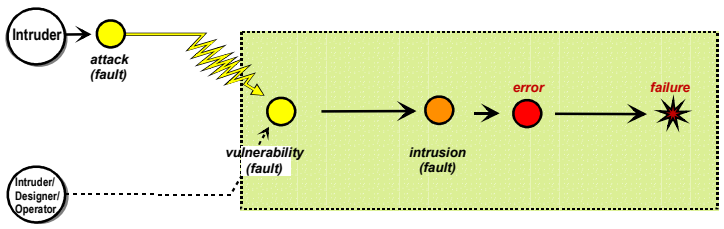
Intrusion Tolerance

Modeling the nature of faults

Attacks, Vulnerabilities, Intrusions

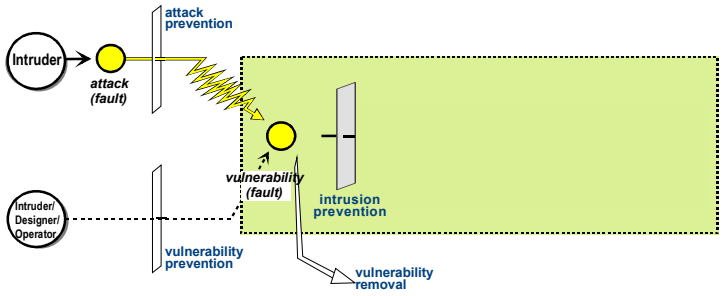
- **Intrusion**
  - an externally induced, intentionally malicious, operational fault, causing an erroneous state in the system
- **An intrusion has two underlying causes:**
- **Vulnerability**
  - malicious or non-malicious weakness in a computing or communication system that can be exploited with malicious intention
- **Attack**
  - malicious intentional fault introduced in a computing or comm's system, with the intent of exploiting a vulnerability in that system
  - without attacks, vulnerabilities are harmless
  - without vulnerabilities, there cannot be successful attacks
- **Hence: attack + vulnerability → intrusion → error → failure**
  - A specialization of the generic *"fault,error,failure"* sequence

Attack-Vulnerability-Intrusion composite fault model:
expressive w.r.t. causes



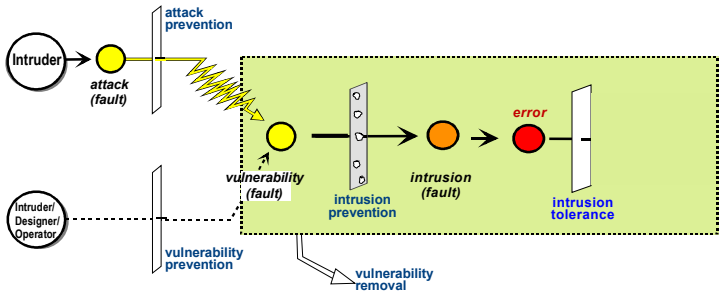
AVI sequence : *attack + vulnerability→ intrusion → error → failure*

AVI Composite fault model:
expressive w.r.t. solutions



- sequence : *attack + vulnerability→ intrusion→ failure*

AVI Composite fault model



- sequence : *attack + vulnerability→ intrusion→ failure*

Intrusion Tolerance

Trust, Trustworthiness

Putting the coverage problem in perspective

Did you say trusted?

- Sometimes components are tamper-proof, others tamper-resistant...
  - Watch-maker syndrome:
    - --- *"Is this watch waterproof?"*
    - --- *"No, it's water-resistant"*
    - --- *"Anyway, I assume that I can swim with it!"*
    - --- *"Well…yes, you can… but i wouldn't trust that very much"*
- How can something trusted be not trustworthy?
  - *Unjustified reliance syndrome:*
    - --- *"I trust Alice"*
    - --- *"Well Bob, you shouldn't, she's not trustworthy"*
- What is the difference? If we separate specification from implementation, and provide notions of *justification* and of *coverage*, all becomes clearer

On Trust and Trustworthiness

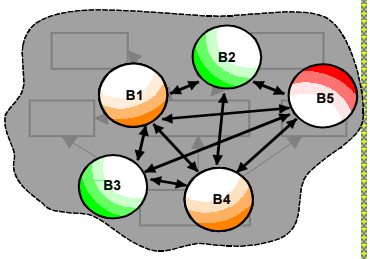
- ***Thou shalt not trust non-trustworthy components!***
  - A trusted component has a set of properties on which another component(s) depend…
- **Trust should be placed to the extent of that component's trustworthiness, the measure in which it meets those properties**
  - trust may have several degrees, quantitatively or qualitatively
  - related not only with security-relat. properties (e.g., timeliness)
  - trust and trustworthiness lead to complementary aspects of the design and verification process
- **when A trusts B, A assumes something about B: Trustworthiness of B measures the coverage of the assumption**
- **... and trustworthiness is never 1 in real systems...**

Tamperproofness and its coverage or "tamper-resistance" not needed

- **Tamperproof**
  - Property of a system/component of being shielded, i.e. whose attack model is that attacks can only be made at the regular interface
  - Coverage of the "tamperproof" assumption may not be perfect, and there can be several degrees of such tamperproofness
- **Example:**
  - *Implementation of a security service using Java Cards to store private keys. We assume J.Cards are tamperproof, and so we argue that they are trustworthy (they will not reveal these keys to an unauthorised party). Hence we can justifiably argue that the service is trusted, with the coverage given by our assumptions, namely, the tamperproofness of JCards*

Building trust

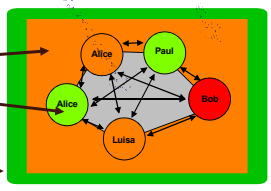
- Component-based approach
- Separation of concerns:
  - higher level algorithms or assertions (e.g., authent/authoriz. logics);
  - infrastructure running them (e.g., procs/servers/comm's)
  - Or:
  - Component builder (trustworthiness)
  - Component user (trust)



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 (Pr_e)
  - set of assumptions (H) about the environment where system will run
  - $Pr_e = Pr(H \mid f)$ *f- any fault*
- operational assumption coverage (Pr_o)
  - the assumptions about how the system/algorithm/mechanism proper (A) will run, under a given set of environmental assumptions
  - $Pr_o = Pr(A \mid H)$

$Pr(A) = Pro \times Pre = Pr(A \mid H) \times Pr(H \mid f)$



Intrusion Tolerance

The substance of Fault Models

The usual path

- If you want efficient/performant solutions to F/T
  - assume controlled failure modes (omissive, fail-silent, etc.)
- If you want to build timely services (even soft RT)
  - assume synchronous models, or at least partially sync
- Some security-related systems take this approach
  - partial synchronous environment
  - well-behaved (e.g. fortress) hosts
  - moderate level of threat in network
- They work, but only to the coverage of the assumptions
  - which must be substantiated
  - else we fall in the "*well-behaved hacker" syndrome*:
    - » *``Hello, I'll be your hacker today, here is the list of what I promise not to do."*
    - » *``Oh thank you! By the way, here are a few additional attacks we would also like you not to attempt."*

Modeling malicious failures

- **What are malicious failures?**
  - how do we model the mind and power of the attacker?
- **Controlled failure assumptions:**
  - fail-controlled models limit what the adversary can do: assume qualitative and quantitative restrictions on compon. failures
  - lead to simpler and more efficient algorithms and systems
  - hard to specify for malicious faults, that brings a coverage problem
    - perhaps best to avoid that pitfall...
- **Arbitrary failure assumptions:**
  - unrestricted failures, limited only to the "possible" failures a component might exhibit (e.g. byzantine)
  - fail-arbitrary models do not limit what the adversary can do, too much…: assumed underlying model (e.g. sync); number of *f*
  - are safe, but normally inefficient

The problem of time and timeliness

- **Can we have secure synchronous (real-time) protocols?**
  - timely behaviour is desirable in pratical secure systems
- **Synchronous models (timed):**
  - solve timed problems, achieve timeliness
  - yield simple algorithms
  - but susceptible to attacks on timing assumptions
  - let alone the difficulty of implementation even in benign settings
    - perhaps best to avoid that pitfall...
- **Asynchronous models (time-free):**
  - resist attacks on timing assumptions
  - efficient probabilistic approaches
  - FLP: no deterministic solution of hard problems e.g. consensus, BA
  - does not solve timed problems (e.g., e-com, stocks)

Where do we go from here?

- **arbitrary failures / asynchrony thread**
  - are safe, but normally inefficient
  - FLP: no deterministic solution of hard problems e.g. consensus, BA
  - does not solve timed problems (e.g., e-com, stocks)
- **controlled failures / synchrony thread**
  - hard to specify for malicious faults, that brings a coverage problem
  - susceptible to attacks on timing assumptions
  - difficulty of implementation of sync. even in benign settings

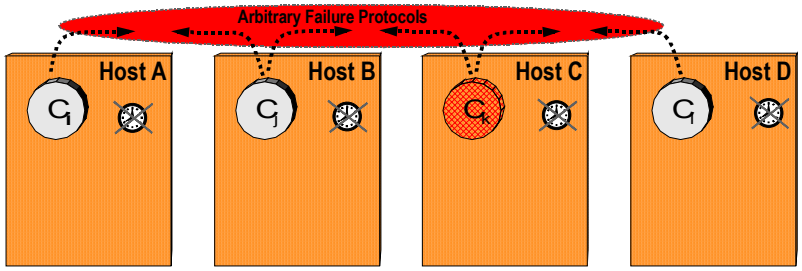
Intrusion Tolerance

Where do we go from here?

- **arbitrary failures / asynchrony thread**
  - are safe, but normally inefficient
  - FLP: no deterministic solution of hard problems e.g. consensus, BA
  - does not solve timed problems (e.g., e-com, stocks)
- controlled failures / synchrony thread
  - hard to specify for malicious faults, that brings a coverage problem
  - susceptible to attacks on timing assumptions
  - let alone the difficulty of implementation even in benign settings

Arbitrary failures / asynchrony thread

- Time-free
- Arbitrary failure environment
- Arbitrary failure resilient protocols



Arbitrary failure / asynchrony assumptions

- OBJECTIVE:
  - solve most non-timed problems with high coverage
- tone down determinism:
  - randomization (Maftia/IBMZurich/Cachin-et-al)
  - semantics (+) - speed (-)
- tone down liveness expectations:
  - sacrifice liveness guarantees (MIT/Castro-Liskov)
  - termination (-) - speed (+)
- use weaker semantics
  - avoid consensus (Cornell/APSS/Schneider-et-al)
  - semantics (-) - termination (+)
- Coverage:
  - very high, but still bound to crucial assumptions, such as number of failures
- Timeliness:
  - none

Where do we go from here?

- arbitrary failures / asynchrony thread
  - are safe, but normally inefficient
  - FLP: no deterministic solution of hard problems e.g. consensus, BA
  - does not solve timed problems (e.g., e-com, stocks)

- **controlled failures / synchrony thread**
  - **hard to specify for malicious faults, that brings a coverage problem**
  - **susceptible to attacks on timing assumptions**
  - **let alone the difficulty of implementation even in benign settings**

Controlled failure assumptions

- **OBJECTIVE:**
  - **solve non-timed problems with high coverage**
- **tone down fault severity:**
  - **hybrid faults (IBMZurich/Cachin-et-al) (Meyer, Pradhan, Walter, Suri)**
  - **fault coverage (+)**
- **enforce hybrid behaviour ("strong" and "weak" components):**
  - **architectural hybridization (MAFTIA/Lisboa)**
  - **speed (+) - termination (+) - semantics (+)**
- **Coverage:**
  - **fair for hybrid fault coverage**
  - **can get very high if bound only to the "strong" components**
  - **still bound to crucial assumptions, such as nr of failures**
- **Timeliness:**
  - **none**

Controlled failures / synchrony assumptions

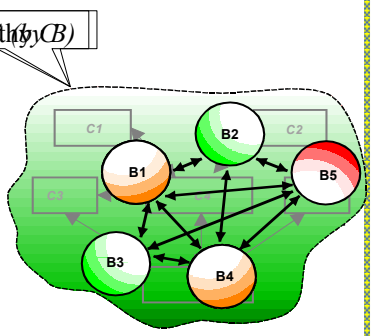
- **OBJECTIVE:**
  - **solve timed and non-timed problems with high coverage**
- **enforce hybrid behaviour w.r.t. time:**
  - **protect crucial time, be indulgent with non-crucial time** (MAFTIA/Lisboa)
- **Real-Time security kernels**
  - **protect time from attackers and other faults** (MAFTIA/Lisboa)
- **indulgent timing assumptions that resist a certain level of threat**
  - **timely computing base** (MAFTIA/Lisboa)
- **Coverage:**
  - can get very high if bound only to the "strong" components
  - still bound to crucial assumptions, such as nr of failures
- **Timeliness:**
  - possible, with "strong" timed components

Modeling and handling malicious failures

- **Intrusion-aware composite fault models**
  - the competitive edge over the hacker
  - AVI: attack-vulnerability-intrusion fault model
- **Combined use of prevention and tolerance**
  - malicious failure universe reduction
  - attack prevention, vulnerability prevention, vulnerability removal, in system architecture subsets and/or functional domains subsets
- **Architectural hybridization**
  - different failure modes for distinct components
  - reduce complexity and increase performance, maintaining coverage
- **Quantifiable assumption coverage**
  - fault forecasting on AVI

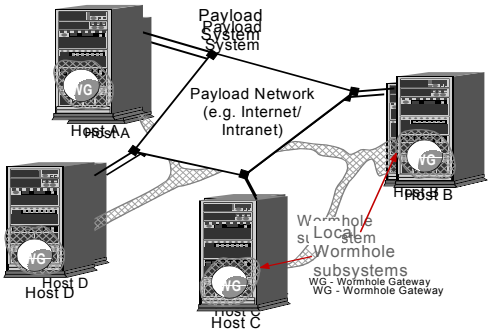
A robust design approach

- **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



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)**



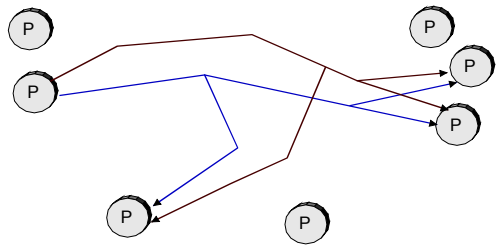
Characterisation of a Wormhole

- 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)

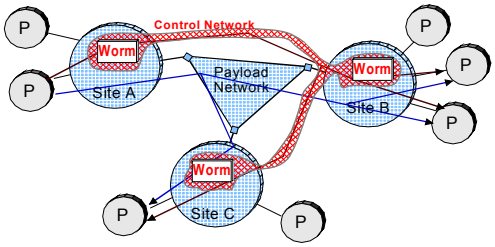
Wormholes seen from inside

- 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



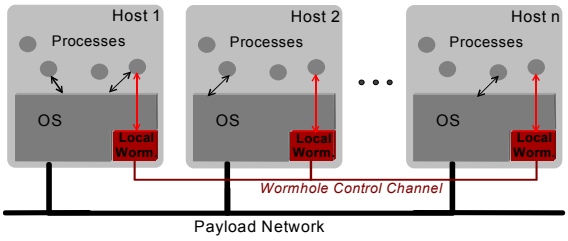
Wormholes seen from inside

- 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

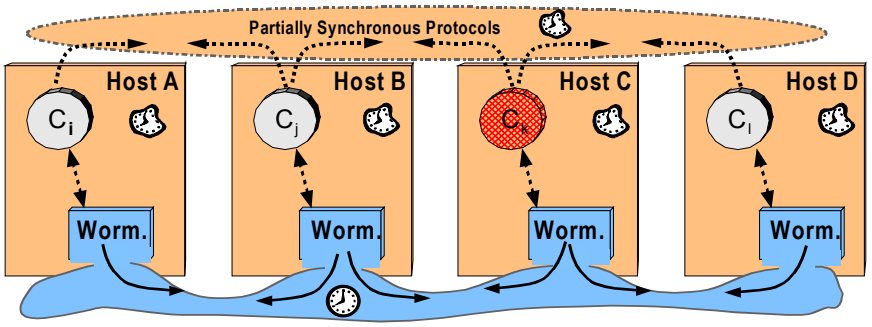


Wormholes seen from inside

- 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

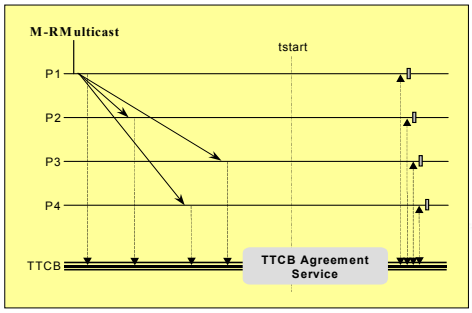


Strategy for timeliness awareness and/or assurance

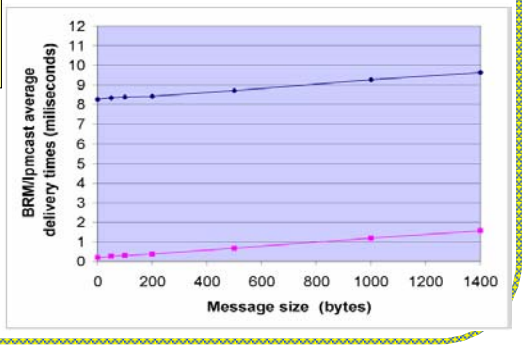


- Fully synchronous, timely
- Partially synchronous, potentially untimely

Byzantine-Reliable Multicast on Timed Model with TTCB

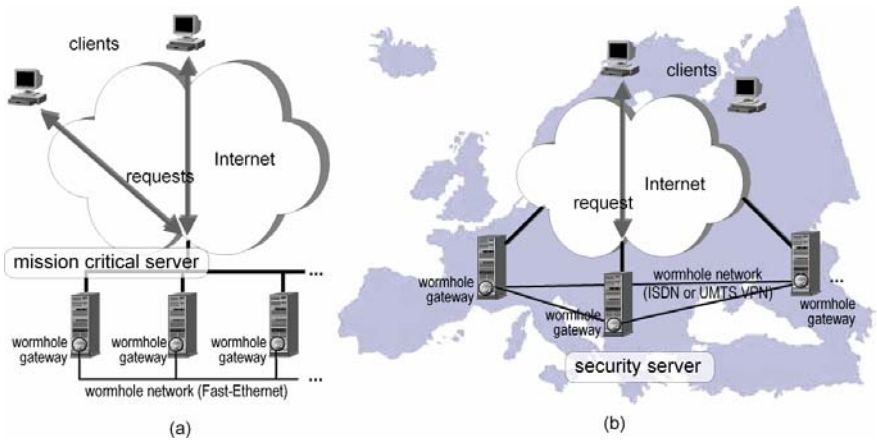


Byzantine Reliable Multicast Protocol (1 Phase)



5-Node Delivery Times

Example of deployment of systems with wormholes



Want to know more about wormholes?

- ***Welcome to the MAFTIA Demos***
- **Wednesday, 14:00-16:00, Golden Gate Room**

Where to find us

- ***Navigators Group***
  http://www.navigators.di.fc.ul.pt
- **Paulo:** *pjv@di.fc.ul.pt*