Resilient Distributed Systems and Algorithms

ReSIST NoE Courseware

Author: Paulo Veríssimo
Editor: Miguel Correia
Univ. of Lisboa Faculty of Sciences
Lisboa - Portugal
http://www.navigators.di.fc.ul.pt
Contents of the course

0- The case for resilience
1- Introduction to fault and intrusion tolerance
2- Resilience building paradigms
3- Models of resilient systems
4- Example resilient systems
The Case for Resilience
Why do systems need resilience?

- **a non-canonical definition of resilience:**
  - “Ability to recover from or adjust easily to misfortune or change.”

- **the case for resilience:**
  1. we want systems to operate through faults and attacks in a seamless manner, in an automatic way
  2. we want systems to endure fact that operating conditions and environments are everyday more uncertain and/or hostile
  3. we want systems to endure their own evolution/reconfig.
  4. we want systems to be deployed in unattended manner

- **intrusion tolerance lets us achieve that**
  2. we want systems to endure fact that operating conditions and environments are everyday more uncertain and/or hostile
  3. we want systems to endure their own evolution/reconfig.
  4. we want systems to be deployed in unattended manner

- **intrusion tolerance is insufficient for that**
  4. we need extra predicates
1. Architecting intrusion-tolerant systems
2. Tolerating Intrusions
3. Handling Attack Severity
4. Resisting Attacks
5. Validating Attacks
Designing for resilience

1. Architecting intrusion-tolerant systems
   - in I/T systems, middleware layers mask failures below, used by upper layers transparently of how tolerance is achieved
   - middleware generally composed of $n$ replicas cooperating through distributed protocols
2. Tolerating Intrusions

- replicas are attacked and corrupted at the measure of the power of threats (attacks, accidents)
- as long as there are sufficient replicas to perform the service correctly, system continues to function
- ... sometimes even without user noticing anything
3. Handling Attack Severity

- expected threats are severe (e.g., malicious intelligence), so protocols should resist to arbitrary faults (i.e., Byzantine)

- necessary quorum for Byzantine resilience to faults is typically $n = 3f + 1$ replicas

- for I/T middleware, goal is to always preserve the number of replicas above that minimum threshold

- can be costly, both in terms of replicas and protocol complexity/ performance
4. Resisting Attacks

- unattended (automatic) system faces resource exhaustion (by faults and attacks) leading to inevitable failure
- threats potentially so intense that this is not an academic possibility: exacerbated by attacker power, common-mode vulnerabilities
- additional defences often required to shrink attackers’ chances and slow down rate of failures preventing resource exhaustion
- e.g., diversity, obfuscation, hybridization, trusted-trustworthy components, rejuvenation
5. Validating attacks

- necessary to study and understand malicious faults in order to validate the fault assumptions underlying the above-mentioned intrusion-tolerant algorithms
- for I/T middleware, this would allow algorithm and system designers to introduce more realistic assumptions and thus increase coverage
- still far from thorough understanding of mechanisms behind attack-vulnerability-intrusion trilogy
Introduction to Fault and Intrusion Tolerance
Brief topics on security & dependability
The failure of computers

- Why do computers fail and what can we do about it?
  
  [ J. Gray]

- Because:
  - All that works, fails
  - We tend to overestimate our HW e SW--- that's called faith]

- So, we had better prevent (failures) than remedy
  - Must do it in a predictable and repeatable way

- Short of faith, we need:
  - a scientific way to quantify, predict, prevent, tolerate, the effect of disturbances that affect the operation of the system
Does not get better with distribution

- A distributed system is the one that prevents you from working because of the failure of a machine that you had never heard of.  

[ L. Lamport]

- Since:
  - Machines fail independently, for a start
  - But they may influence each other,
  - They communicate through unreliable networks, with unpredictable delays

- ...gathering machines renders the situation worse:
  - The reliability (<1) of a system is the product of the individual component reliabilities, for independent component failures
  - \[ R(10 @ 0.99) = 0.99^{10} = 0.90 \]
  - \[ R(10 @ 0.90) = 0.90^{10} = 0.35 \]
Can get much worse with malicious failures

- Failures are no longer independent
- Failures become more severe
- Fault models become less representative

... Hackers don’t like stochastics ...
Intrusion Tolerance
What is Intrusion Tolerance?

• 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, with a measurable probability

• 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 preliminary observations...
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 not be 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 a notion of coverage, all becomes clearer
Trust, Trustworthiness

- **Trust**
  - the accepted dependence of a component, on a set of properties (functional and/or non-functional) of another component, subsystem or system
    - a trusted component has a set of properties that are relied upon by another component (or components).
    - if A trusts B, then A accepts that a violation in those properties of B might compromise the correct operation of A

- **Trustworthiness**
  - the measure in which a component, subsystem or system meets a set of properties (functional and/or non-functional)
    - trustworthiness of B measures the coverage of the trust of A
Trusted vs. Trustworthy

• Thou shalt not trust non-trustworthy components!
• B is Trustworthy in the measure of the coverage with which its assumed properties are met... and coverage is never 1 in real systems...
• B should be Trusted only to the extent of its trustworthiness
  - 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
• we should talk about trusted-trustworthy components
Intrusion Tolerance
terminology and concepts

Fault Models
Methodologies
Error processing
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 comm’s 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

**interesting corollaries:**

- 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

AVI sequence: attack + vulnerability → intrusion → error → failure
Intrusion Tolerance

terminology and concepts

Fault Models
Methodologies
Error processing
Achieving trustworthiness w.r.t. malicious faults (the classical ways...)

- **Attack prevention**
  - Ensuring attacks do not take place against certain components

- **Attack removal**
  - Taking measures to discontinue attacks that took place

- **Vulnerability prevention**
  - Ensuring vulnerabilities do not develop in certain components

- **Vulnerability removal**
  - Eliminating vulnerabilities in certain components (e.g. bugs)

**INTRUSION PREVENTION**
AVI Composite fault model

sequence: attack + vulnerability → intrusion → failure
AVI Composite fault model

\[ \text{sequence: } \text{attack + vulnerability} \rightarrow \text{intrusion} \rightarrow \text{failure} \]
Intrusion Tolerance

terminology and concepts

Fault Models
Methodologies
Error processing
Error processing at work

- backward recovery
- forward recovery
- error masking
ID: Error detection or fault diagnosis?

• classical IDS have two facets under intrusion tolerance
  - detecting errors as per the security policy specification
  - diagnosing faults as per the system fault model

• consider the following example:
  - Organization A has an intranet with an extranet connected to the public Internet. It is fit with an IDS
  - the IDS detects a port scan against an internal host, coming from the intranet
  - the IDS detects a port scan against one of the extranet hosts, coming from the Internet
  - what is the difference?
Intrusion Forecasting
Approaches

- Fault injection
- Static vulnerability analyzers
- Run-time prevention mechanisms
- Vulnerability scanners
- Fuzzers
- Attack injection - Using Attacks to Find Vulnerabilities
Using Attacks to Find Vulnerabilities

- Composite fault model AVI (Attack, Vulnerability, and Intrusion)

1. Generate various attacks
2. Look for errors / failures
3. Find the correspondent vulnerability for that particular attack
Attack Injection Tool

- Architecture

Target Protocol Specification

XML spec

Target System

Attack Injector

Monitor

attack
response
synchronization

execution data
Example Intrusion-Tolerant Networks and Architectures
Trusted-Third-Party Security Server
• Intrusion prevention device: prevents attacks on inside machines
• Coverage: semantics of firewall functions, resilience of bastions
• End-to-end problem: are all internal network guys, good guys?
Client-Server with Intrusion Tolerant Servers

Data Network

Client PC or WS

Intrusion-Tolerant Application Servers
• Intrusion prevention device: enforces confidentiality, integrity (authenticity)
• Coverage: tunnelling method, resilience of gateway
• End-to-end problem: are all intranet guys, good guys?
Authentication, signatures, MACs

- Intrusion prevention device: enforces authenticity, integrity
- Coverage: signature/authentication method
- End-to-end problem: who am I authenticating? me or my PC?
Trusted Third Party (TTP) protocols

- Intrusion tolerance device: error processing/masking
- Coverage: semantics of protocol functions, underlying model assumptions, resilience of TTP
Communication and agreement protocols

- Intrusion tolerance device: error processing or masking (3f+1, 2f+1, f+2)
- Coverage: semantics of protocol functions, underlying model assumptions
Threshold cryptography

- Intrusion tolerance device: error processing/masking (f+1 out of n)
- Coverage: crypto semantics, brute-force attack resilience, underlying model assumptions
2

Resilience Building Paradigms
Resilience building paradigms

- Intrusion Detection
- Byzantine Failure Detection
- Trusted Third Parties
- Threshold cryptography
- Secret sharing
- Byzantine Reliable Broadcast
- Byzantine agreement
- Byzantine Consensus and Atomic Bcast
- Byzantine State Machine Replication

- Byzantine Quorums
- Fragmentation
- Randomisation
- Indulgence
- Separate exec/agreement
- Wormholes
- Exhaustion Safety
- Reactive/Proactive recovery
- Diversity and obfuscation
- Proactive resilience
Resilience building paradigms

- Intrusion Detection
Detection mechanisms

- consider system activity specified by patterns
- anomaly detection
  - looks for deviation from NORMAL ACTIVITY PATTERNS
- misuse detection
  - looks for existence of ABNORMAL ACTIVITY PATTERNS
- we can have hybrids
- Quality of Service
  - false alarm rate
  - omitted alarm rate
Intrusion vs. Failure Detection

- intrusion detection bears resemblance to ‘failure detection’ in distributed systems
- only in the ‘error detection’ facet
- allows elegant automation of recovery procedures if embedded in intrusion-tolerant protocols
Resilience building paradigms

- Self-enforcing vs. Trusted Third Party protocols
Self-enforcing protocols

- Correct behaviour achieved by protocol participants alone
- They must build trust during protocol execution without trusting each other initially, and some maybe being malicious (e.g., by voting, k-out-of-n)
Trusted-Third-Party protocols

- Based on an apriori trusted component (TTP)
- TTP may be single point of failure
- adjudicated
  - Acting a posteriori if necessary to recover from errors
- arbitrated
  - Correct behaviour guaranteed during execution, errors prevented by arbiter
- certified
  - Correct behaviour guaranteed prior to execution through credentials supplied which limit participants misbehaviour during execution (errors prevented)
Resilience building paradigms

- Threshold cryptography
- Secret sharing
- Proactive secret sharing
Threshold cryptography and secret sharing

- "Intrusion-tolerant" cryptography
- Given N processes each holding part of crypto secret

- **Secret sharing:**
  - Example: a shared secret key
  - Any k-out-of-N processes combine their shares and reconstruct secret s
  - Any f=k-1 colluding or intruded processes cannot reconstruct s

- **Function sharing:**
  - Example: a threshold signature
  - k processes together execute function F
  - f=k-1 colluding or intruded processes cannot execute F
Proactive secret sharing

- A process cannot know whether its share is “good”
- If one share is corrupted the secret is not reconstructed

- **Proactive secret sharing**
  - A period $T_f$ is assumed as an estimate of time for $f+1=k$ failures to be produced, e.g., to corrupt $k$ processes
  - (these $k$ processes would be able to get the secret)
  - Every $T_{ss} < T_f$, protocol recalculates the shares (reconstructs) without changing the secret
Resilience building paradigms

• Byzantine Reliable Broadcast/Multicast
• Byzantine agreement
Basic failure modes

• Processes can fail in a Byzantine way:
  - Crash, disobey the protocol, send contradictory messages, collude with other malicious processes,...

• Network:
  - Can corrupt packets (due to accidental faults)
  - An attacker can modify, delete, and introduce messages in the network
Byzantine Reliable multicast

- A reliable multicast protocol is defined formally in terms of the following properties:

  - **Validity:**
    - If a correct process multicasts (sends) a message $M$ then some correct process in $\text{group}(M)$ eventually delivers $M$.

  - **Agreement:**
    - If a correct process delivers a message $M$ then all correct processes in $\text{group}(M)$ eventually deliver $M$.

  - **Integrity:**
    - For any message $M$, every correct process $p$ delivers $M$ at most once and only if $p$ is in $\text{group}(M)$, and if $\text{sender}(M)$ is correct then $M$ was previously multicast by $\text{sender}(M)$.
Resilience building paradigms

- Byzantine Consensus
- Atomic Broadcast
Byzantine Consensus properties

• **Validity**
  - If all correct processes propose the same value \( v \), then any correct process that decides, decides \( v \)

• **Agreement**
  - No two correct processes decide differently

• **Termination**
  - Every correct process eventually decides

• With Byzantine failures, Validity makes little sense
• Vector consensus improves the situation

• **Consensus is equivalent to atomic broadcast**
Byzantine Vector Consensus properties

- **Validity**
  - Every correct process decides on a vector vect of size n such that:
  - 1. For every $1 \leq i \leq n$, if process $p_i$ is correct, then $\text{vect}[i]$ is either the initial value of $p_i$ or the value bottom
  - 2. At least $f+1$ elements of the vector vect are the initial values of correct processes.

- **Agreement**
  - No two correct processes decide differently

- **Termination**
  - Every correct process eventually decides
• **Validity**
  - If a correct processor multicasts a message $M$, then some correct processor eventually delivers $M$.

• **Agreement**
  - If a correct processor delivers a message $M$, then all correct processors eventually deliver $M$.

• **Integrity**
  - For any message $M$, every correct processor $p$ delivers $M$ at most once, and if $\text{sender}(M)$ is correct then $M$ was previously broadcast by $\text{sender}(M)$.

• **Total order**
  - If two correct processors deliver two messages $M_1$ and $M_2$ then both processors deliver the two messages in the same order.

• **Pillar of state machine replication**
Resilience building paradigms

• Byzantine State Machine Replication
Byzantine State Machine Replication

• **Rules:**
  - they execute atomic commands, change state and produce outputs
  - commands are deterministic

• **If:**
  - servers start in same state
  - execute same sequence of inputs in same order

• **Then,**
  - all follow same sequence of state/outputs
Byzantine State Machine Replication

- input requirements:
  - commands delivered by Byzantine atomic broadcast protocol
- Failures of servers can be arbitrary
- given N number of servers, maximum number of servers that can fail is: 
  \[ f = \left\lfloor \frac{N-1}{3} \right\rfloor \]
- or in other words: 
  \[ N \geq 3f + 1 \]

- this limit is actually imposed by the protocol used to disseminate messages (ABCAST)
  - ex: N=4 servers tolerate f=1 corrupt; N=7 tolerates f=2
State of play in intrusion-tolerant paradigms

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

- Where do we go from here?
  - i.e., how to address these problems?
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/Zhou, Schneider, Rennesse)
  - use quorums (Alvisi, Malki, Reiter)
  - use fragmentation (Fraga, Powell, Deswarte)
  - semantics (-) - termination (+)

- **Coverage:**
  - very high, but still bound to crucial assumptions, such as number of failures

- **Timeliness:**
  - none
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 w/ Wormholes (U.Lisboa)
  - speed (+) - termination (+) - semantics (+)
  - fault coverage (+)

• Coverage:
  - fair for hybrid fault coverage
  - can get very high if bound to the “strong” components
  - still bound to crucial assumptions, such as nr of failures

• Timeliness:
  - none
Resilience to attacks
Resilience building paradigms

- Exhaustion safety
Taking long detours...

- **OBJECTIVE +**:  
  - keep systems working long enough regardless of paradigm used  
    (non-timed problems, arbitrary failures / asynchrony thread)

- ensuring enough replicas
- using diversity and obfuscation

- **OBJECTIVE ++**:  
  - keep systems working in a perpetual manner

- reactive or proactive recovery (e.g., rejuvenation, refreshing)
Detours may lead to dead ends...

- f fault-tolerance means keeping at least n-f correct nodes.
- Resource exhaustion: violation of a resource assumption (e.g., f+1 nodes fail), which may lead to failure
- A system is exhausttion-safe if resource exhaustion never happens.

Becomes a crucial problem with malicious faults
To Be or Not to Be Exhaustion-Safe

- not executing
- immune to exhaustion-failures
- vulnerable to exhaustion-failures

exhaustion-safe

\[ A_{t_{\text{start}}} \quad A_{t_{\text{end}}} \quad A_{t_{\text{exhaust}}} \]

non exhaustion-safe

\[ A_{t_{\text{start}}} \quad A_{t_{\text{exhaust}}} \quad A_{t_{\text{end}}} \]
To Be or Not to Be Exhaustion-Safe

- An $f$ fault-tolerant distributed system is exhaustion-safe if it terminates before $f+1$ faults being produced

- Obvious?
  - what if it never terminates?
  - what if we don’t know when it will terminate?
Resilience building paradigms

- Reactive/Proactive recovery
- Diversity and obfuscation
- Proactive resilience
Async Proactive Recovery

- How to guarantee that rejuvenations always terminate before resource exhaustion?
  - Rejuvenation start instant may be delayed.
  - Rejuvenation actions may be delayed.
  - These delays may be enforced by a malicious adversary!

- Async proactive recovery does not guarantee exhaustion-safety, namely, in a malicious environment:

- Impossibility of exhaustion-safe asynchronous distributed systems (w/ or w/o proactive recovery)

[Sousa, Veríssimo, Neves]
[Cachin et al., Zhou et al., Castro et al.]
Proactive resilience

- Design proactive recovery under hybrid models:
  - proactive recovery is useful to postpone $t_{\text{exhaust}}$ as long as it has timeliness guarantees ($t_{\text{recoy}} < t_{\text{exhaust}}$)
  - cannot be done in homogeneous async. systems
  - should be done in homogeneous sync. systems
  - use hybrid systems:
    - combine async payload system with sync/parsync proactive recovery subsystem
    - under a Wormhole model (hybrid distributed systems model)
Limitations of current I/T paradigms

Some practical examples
Limitations of I/T paradigms

• Resource exhaustion unnoticed
  or
• attackers work in real time!
Classical Model - Async System

Async

Alice  Paul  Bob

Trent

N
Burn like a candle or....
Burn like a match?

Classical Model vs. Reality

- Alice
- Paul
- Trent
- Bob

external fault/attack production timeline (physical)

internal system evolution timeline (logical)
Asynchronous Proactive Recovery

- **Goal:** to constantly postpone $t_{\text{exhaust}}$ through periodic rejuvenation.
  - e.g., periodic rejuvenation of secret keys, OS code, etc.

- A system is exhaustion-safe only if rejuvenations are always **terminated** before exhaustion.
Classical Model - Async system with unaccounted for synchrony

Async

Alice

Paul

Bob

Trent

FT Protocol

Sync $N > N' < N$

$\phi$

NOT Exhaustion-safe!

NOT Exhaustion-safe!
The Problem

- **Goal recall:** to constantly postpone $t_{\text{exhaust}}$ through periodic rejuvenation.
  - e.g., periodic rejuvenation of OS code.

Resources are exhausted ... system correctness can be compromised.
The case for hybrid dist. sys. models

- in an asynchronous system, any synchronous behaviour must be encapsulated
Limitations of I/T paradigms

- Homogenous models and hidden assumptions
- or
- attackers pick the weakest link!
Classical Model - Async System

Async

Alice  Paul  Bob  Trent
Asynchronous Proactive Secret Sharing

- **System Model:**
  - asynchronous model, malicious adversary.
  - private key shared by servers using threshold cryptography.
  - shares are periodically refreshed through an **asynchronous** proactive secret sharing protocol
  - key is compromised if an adversary collects sufficient shares in the **interval between successive executions** of the protocol
  - recovery instants determined by internal system clock
Classical Model - Async system with hidden synchrony assumptions
The problem

• Algorithmic Assumptions:
  - n servers share private key, f+1 shares sufficient to recover it
  - at most f servers “are compromised at any time”.
  - “does not rule out learning f+1 shares one at a time” (mobile virus attack)

• safety of an asynchronous system depends on non-substantiated synchrony assumptions:
  - clocks with bounded rate of deviation to real-time
  - capacity of performing periodic (timely) executions

• these assumptions can be violated:
  - in the assumed asynchronous environment
  - and/or by a malicious adversary.
From Theory to Practice

• An attack that compromises safety:
  - Two adversaries: ADV1 and ADV2.

• Step 1: ADV1 performs a mobile virus attack against f+1 servers
  - slows the clock rate of each server.

• Step 2: ADV1 temporarily cuts off the links between the f+1 servers and the rest of the system.

• Important Note: ADV1 actions simply enforce a behavior that can occur in any fault-free async system.
• An attack that compromises safety:

• Step 3: ADV2 performs a mobile virus attack against the same f+1 servers
  - learns, one by one, f+1 private key shares.
  - no rejuvenation occurs in between because in step 1 clocks are made as slow as needed.

• Step 4: ADV2 discloses private key by combining the f+1 shares.
• Example with $n=4$, $f=1$
From Theory to Practice

- Example with $n=4$, $f=1$

ADV1 slows A clock
From Theory to Practice

- Example with $n=4$, $f=1$

A B C D

ADV1 slows B clock
From Theory to Practice

- Example with $n=4$, $f=1$

ADV1 cuts off connection between A, B and C, D
From Theory to Practice

• Example with n=4, f=1

ADV2 gets A share

---

s25  s25  s50  s50
A     B     C     D

99
From Theory to Practice

- Example with \( n=4, f=1 \)

A share + B share = private key
From Theory to Practice

- Example with $n=4$, $f=1$
The case for hybrid dist. sys. models

- in an asynchronous system, all timing assumptions must be encapsulated

Classical Model - Correct FT Async system

Async

All $t$ factored out e.g. to FD oracles
Findings

• Current state-of-the-art with homogenous models does not allow to construct exhaustion-safe distributed systems, specially in face of arbitrary/malicious faults:

• Sync systems are vulnerable:
  - timing failures.

• Async systems are vulnerable:
  - max number of faults + unbounded execution time.

• Async systems with async proactive recovery are vulnerable:
  - max number of faults + unbounded rejuvenation period.
Proactive Resilience at work in Wormhole (hybrid) models

• define the number of faults between rejuvenations
• compute rejuvenation period
• execute recovery:
  - timely triggered
  - executed in bounded time
3

Models of Resilient Systems
Intrusion Tolerance

strategies
Asynchronous Fail-uncontrolled strategy

- Time-free
- Arbitrary failure environment
- Arbitrary-failure resilient protocols
- Used e.g. with: probabilistic Byzantine-agreement or consensus protocols
- Impossibility results for deterministic protocols, and for any timed operation
Partially-synchronous Fail-controlled strategy

- Timed, partially synchronous
- Non-Arbitrary failure environment and protocols
- Used e.g. with: classical reliable multicast and atomic broadcast
- Problem of coverage of assumptions
Recall: Where do we go from here?

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

- controlled failures / synchrony thread
  - hard to specify for malicious faults and that brings a coverage problem
  - susceptible to attacks on timing assumptions
  - difficulty of implementation of sync. even in benign settings
• **OBJECTIVE:**
  - solve most non-timed problems with highest possible coverage

• **tone down determinism** (e.g., randomisation)
• **tone down liveness expectations** (e.g., indulgence)
• **use weaker semantics** (e.g., thresholds, quorums)
• **tone down allowed fault severity** (e.g., hybrid faults)
• **tone down asynchrony** (e.g., parsync protocols, FDs)

• **OBJECTIVE:**
  - solve timed problems with highest possible coverage

• **tone down asynchrony** (e.g., sync/parsync protocols)

Recall: Taking **detours**...
Intuition behind hybrid models
Take time/synchrony facet

• OBSERVATION [Veríssimo and Casimiro. The Timely Computing Base model and architecture. DI/FCUL TR-99-2, IEEE TOCS 2002]:
  “synchronism is not an invariant property of systems”

• degree of synchronism varies in the time dimension:
  - during the timeline of their execution, systems become faster or slower, actions have greater or smaller bounds

• it also varies with the part of the system being considered, that is, in the space dimension:
  - some components are more predictable and/or faster than others, actions performed in or amongst the former have better defined and/or smaller bounds
Take time/synchrony facet


(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)
Take time/synchrony facet

How does it work under homogeneous models?!

- expecting/enforcing
- eventual/perpetual
- discrete

- expecting
- eventual
- continuous

Partial synchrony

(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)

Homogeneous distr. sys. models

- synchronous/secure
- asynchronous/insecure
Homogeneous distr. sys. models

- synchronous/secure
- asynchronous/insecure
Advanced modelling concepts for I/T systems
Some principles

• gain competitive edge over the hacker
• intrusion-aware composite fault & intrusion models
  - AVI: attack-vulnerability-intrusion fault model
• architecturally hybrid failure assumptions
  - different failure modes for distinct components
  - increase performance maintaining coverage
• combined use of prevention and tolerance
  - vulnerability prevention/removal: trusted-trustworthy components by construction
  - attack prevention: malicious failure universe reduction
• seek quantifiable assumption coverage
  - attack/vulnerability forecasting (on AVI)
Advanced modelling for I/T systems

1. Recursive building of trust and trustworthiness
   - Trusted-trustworthy subsystems out of non-trustworthy components

2. Architectures with hybrid trustworthiness
   - where non-trustworthy AND trustworthy components co-exist

3. Hybrid distributed systems models
   - distributed systems with loci of different behaviour
Hybrid distributed systems models
Shortcuts i.s.o detours

• Rendering the solution simpler
  without changing the problem!

• Architectural hybridization

• Wormholes model

In
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, which allow achieving predictability (the required "hard properties")
Theoretical underpinnings: system model
Architect. hybrid distr. sys. models
Architect. hybrid distr. sys. models

Any-synchrony/security system P

Any-synchrony/security system W
Practical insight
Practical uses of Wormholes

- how they assist protocols time-wise:
  - determine useful facts about time (be sure it executed something on time; measure a duration; determine it was late doing something)
  - acts as a **checkpoint** that time-free participants have to synchronise with and this limits their potential for timing uncertainty
Practical uses of Wormholes

- how they assist protocols security-wise:
  - protocol participants exchange messages in a world full of threats, some of them may even be malicious and cheat
  - there is an oracle that correct participants trust to get info, and/or a channel that they can use to get in touch with each other, even for rare moments
  - acts as a checkpoint that malicious participants have to synchronise with, and this limits their potential for Byzantine actions
An example Wormhole: Trusted Timely Computing Base (TTxCB)

• Properties and services:
  - trusted and timely execution; trusted timing failure detection
  - secure (can only fail by crashing)
  - real-time (capable of timely behaviour)
  - local authentication
  - agreement on a fixed sized block of data (TBA)
  - globally meaningful timestamps

• TTCB can be seen as a distributed security kernel:
  - provides a minimal set of trusted and timely services
  - correct processes can interact securely with the TTCB
  - assists execution of fault/intrusion-tolerant algorithms
  - provides a trusted environment for user-defined kernel functions
Wormholes model in action
Example of deployment of systems with wormholes
Theoretical underpinnings: algorithm design
Designing algorithms with wormholes
(aka hybrid distributed systems models)

Assume basic system P model, e.g. asynch. and Byzantine failures

Postulate existence of components (W) on a different set of assumptions, e.g.:
- failure detector oracle
- set of fast(er) or synch. channels

• Proof correct conditional to truthfulness of assumptions.
• What if assumptions cannot be substantiated?
  • i.e. they do not represent physical reality?

Design your P algorithms and prove them correct

Any-synchrony/security system P

Any-synchrony/security system W
Designing algorithms with wormholes
(aka hybrid distributed systems models)

Assumptions substantiated by architectural hybridization

- Reiterate design, now of system W
- Assume basic system W model, e.g. synch. and crash failure

"Main" or payload subsystem
Design system W's architect/algorithms to provide properties postulated earlier for these components, e.g.: - failure detector oracle - set of fast(er) /synch. channels

Prove them correct

Proof correct conditional to truthfulness of assumptions.

Any-synchrony/ security system P

Any-synchrony/ security system W
Does it work!? 

some proof-of-concept prototypes
Proof-of-concept systems with wormholes

Any-synchrony/security system P

Any-synchrony/security system W
Proof-of-concept: COTS-based TTCB Reference Architecture

Proof-of-concept: Hardware-based Wormholes

- **Connectivity:**
  - Wireless WiFi, Bluetooth
  - Wired RS-232, USB2, Ethernet
Proof-of-concept systems with wormholes

Any-synchrony/security system P

Any-synchrony/security system W
• TTCB is a distributed real-time and security kernel that provides a minimal set of trusted and timely services, such as
  - failure detection
  - local authentication
  - agreement on a fixed sized block of data (TBA)
  - trustworthy global timestamps and random numbers
Weaker wormholes

• Wormholes can be any distributed subsystem/component that follows different assumptions from "main" (payload) system:
  - watchdog
  - crypto chip
  - sync or parSync set of channels
  - timely execution monitor

• There can be more than one wormhole subsystem
• Wormhole subsystems can be constructed as fault or intrusion-tolerant subsystems
Proof-of-concept systems with wormholes
Fault/Intrusion-tolerant wormholes

- Close the “lid”, you now have a trustworthy Wormhole

- Assume Byzantine failures in Wormhole realm

- Use Byzantine resilient algorithms to implement Wormholes services

Any-synchrony/security system P
Byzantine on failure system W
Hybrid models/architectures more complex than homogeneous, when to use them?
Take time/synchrony facet

(Verissimo et al, Fetzer et al, LeLann et al, Castro et al, Raynal et al, Aguilera et al, Friedman et al, Baldoni et al, etc.)

- expecting/enforcing
  - eventual/perpetual
  - discrete

- expecting
  - eventual
  - continuous

(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)
- How to enforce perpetual, discrete behaviour?
- How to get synchrony out of asynchrony?

(Dolev et al, Dwork et al, Chandra et al, Cristian et al, etc.)
Review of
Strategies for construction
of I/T subsystems
Recursive use of F. Prevention and F. Tolerance

- The TTP protocol revisited
- Work at subsystem level to achieve justifiable behaviour
- Architectural hybridation w.r.t. failure assumptions
Strategies for construction of I/T subsystems

- Arbitrary model - no assumptions
- High coverage - very little to “cover”
Strategies for construction of I/T subsystems

- Fail-controlled model -- unjustified environment assumptions
- Fair coverage - no enforcement
- Fail-controlled model - little environment assumptions; justified component assumptions
- High coverage - enforced by Local or Distrib. Trusted Comp.
Byzantine protocols on asynchronous fail-uncontrolled models
Bracha's binary consensus **(local coin)**

- $i_p$ is set to the value proposed by the process initially.
- $(d, v)$ is a special value used to try to decide $v$.
- round($k$) by process $p$.

1. Broadcast($i_p$) and wait for $n-f$ valid messages.
   - $i_p$ = majority of the values in valid messages.
2. Broadcast($i_p$) and wait for $n-f$ valid messages.
   - If more than $n/2$ of the messages have the same value, then $i_p = (d, v)$.
3. Broadcast($i_p$) and wait for $n-f$ valid messages.
   - If there are $n-f (d,v)$ messages then decide $v$.
   - If there are at least $n-2f (d,v)$ messages then $i_p = v$
   - Else $i_p = 1$ or $0$ with probability $1/2$.

   If all propose the same terminates in 1 round.
   Expected number of rounds $= 2^{n-f}$.

A stack of protocols

- Correia, Neves, Veríssimo
  *From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures*
  *Computer Journal, Jan. 2006*
Function Multi Valencia Consensus (v, cid)

INITIALIZATION:
1: INIT-delivered; \( i \leftarrow 0; \)
2: activate task (T1, T2);

TASK T1:
3: R_Broadcast ((INIT, v, cid, i));
4: wait until (at least \( n - f \) INIT messages have been delivered);
5: \( \forall j: \text{if} ((\text{INIT}, v_j, \text{cid}, j) \text{ has been delivered}) \text{ then } V_{i,j} \leftarrow v_j; \text{ else } V_{i,j} \leftarrow \bot; \)
6: \( \text{if} (\exists 1 \leq i \leq \#d(V_i) \geq (n - 2f)) \text{ then} \)
7: \( w_i \leftarrow v; \)
8: else
9: \( w_i \leftarrow \bot; \)
10: R_Broadcast ((VECT, w_i, V_{i,j}, cid, i));
11: wait until (at least \( n - f \) valid messages (VECT, w_j, V_{i,j}, cid, j) have been delivered);
12: \( \forall j: \text{if} ((\text{VECT}, w_j, V_{i,j}, \text{cid}, j) \text{ has been delivered}) \text{ then } W_{i,j} \leftarrow w_j; \text{ else } W_{i,j} \leftarrow \bot; \)
13: \( \text{if} (\forall j \neq k W_{i,j} \neq W_{i,k} \Rightarrow W_{i,j} = \bot \text{ or } W_{i,k} = \bot \text{ and } (\exists a: \#d(W_i) \geq (n - 2f)) \text{ then} \)
14: \( b_i \leftarrow 1; \)
15: else
16: \( b_i \leftarrow 0; \)
17: \( e_i \leftarrow \text{BConsensus}(b_i, \text{cid}); \)
18: \( \text{if} (e_i = 0) \text{ then} \)
19: return \( \bot; \)
20: wait until (at least \( n - 2f \) valid messages (VECT, v_j, V_{i,j}, cid, j) with \( v_j = v \) have been delivered);
21: return \( v; \)

TASK T2:
22: \( \text{when} \ m_i = (\text{INIT}, v_j, \text{cid}, j) \text{ is delivered do} \)
23: \( \text{INIT-delivered}_i \leftarrow \text{INIT-delivered}_i \cup \{m_i\}; \)
Vector consensus

• VC1 Vector validity: Every correct process that decides, decides on a vector $V$ of size $n$:
  - $\forall p_i :$ if $p_i$ is correct, then either $V[i]$ is the value proposed by $p_i$ or $\bot$;
  - at least $(f + 1)$ elements of $V$ were proposed by correct processes.

• VC2 Agreement: No two correct processes decide differently.

• VC3 Termination: Every correct process eventually decides.
Vector consensus

Function Vector Consensus (v_i, vcid)

1: r_i ← 0;
2: R_Broadcast ( 〈VC_INIT, v_i, vcid, i〉 );
3: repeat
4:   wait until (at least \(n - f + r_i\) VC_INIT messages have been delivered);
5:   \(\forall j:\ if\ ( 〈VC_INIT, v_j, vcid, j〉\ has\ been\ delivered)\ then\ W_i[j] ← v_j;\ else\ W_i[j] ← \perp;\)
6:   V_i ← M_V Consensus (W_i, (vcid.r_i));
7:   r_i ← r_i + 1;
8: until (V_i \neq \perp);
9: return V_i;
Atomic broadcast

- **AB1 Validity**: If a correct process broadcasts a message $M$, then some correct process eventually delivers $M$.
- **AB2 Agreement**: If a correct process delivers a message $M$, then all correct processes eventually deliver $M$.
- **AB3 Integrity**: For any identifier $ID$, every correct process $p$ delivers at most one message $M$ with identifier $ID$, and if $\text{sender}(M)$ is correct then $M$ was previously broadcast by $\text{sender}(M)$.
- **AB4 Total order**: If two correct processes deliver two messages $M_1$ and $M_2$ then both processes deliver the two messages in the same order.
Atomic broadcast

INITIALIZATION:
1: $R_{delivered_i} \leftarrow 0$; \hspace{1cm} \{messages delivered by the reliable broadcast protocol\}
2: $\text{aid}_i \leftarrow 0$; \hspace{1cm} \{atomic broadcast identifier\}
3: $\text{num}_i \leftarrow 0$; \hspace{1cm} \{message number\}
4: activate task(T1,T2);

WHEN Procedure $A_{Broadcast}(m)$ is called do
5: $R_{Broadcast}((A_{MSG}, \text{num}_i, m, i))$;
6: $\text{num}_i \leftarrow \text{num}_i + 1$;

TASK T1:
7: when ($R_{delivered_i} \neq 0$) do
8: $H_i \leftarrow \{\text{hashes of the messages in } R_{delivered_i}\}$;
9: $X_i \leftarrow \text{Vector.Consensus}(H_i, \text{aid}_i)$;
10: wait until (all messages with hash in $f + 1$ or more cells in vector $X_i$ are in $R_{delivered_i}$);
11: $A_{deliver_i} \leftarrow \{\text{all messages with hash in } f + 1 \text{ or more cells in vector } X_i\}$;
12: atomically deliver messages in $A_{deliver_i}$ in a deterministic order;
13: $R_{delivered_i} \leftarrow R_{delivered_i} \setminus A_{deliver_i}$;
14: $\text{aid}_i \leftarrow \text{aid}_i + 1$;

TASK T2:
15: when $A_{MSG}$ is delivered by the reliable broadcast protocol do
16: $R_{delivered_i} \leftarrow R_{delivered_i} \cup \{(A_{MSG}, \text{num}_i, m, i)\}$;
Byzantine protocols on hybrid distributed systems models
Byzantine-Resilient Reliable Multicast

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
Basic failure modes

- **Processes can fail in a Byzantine way:**
  - Crash, disobey the protocol, send contradictory messages, collude with other malicious processes,…

- **Network:**
  - Can corrupt packets (due to accidental faults)
  - An attacker can modify, delete, and introduce messages in the network
TTCB services

• The reliable multicast protocol uses only three TTCB services:
  • Local authentication service
  • Trusted block agreement
  • Trusted absolute timestamping
• A process makes two operations:
  - propose, decide
  - this works with “small” blocks of data

• agreement is defined by (elist, tstart, decision)
  - elist: list of processes involved
  - tstart: instant when the TTCB stops accepting proposals
  - decision = TTCB_TBA_RMULTICAST; returns:
    • value proposed by 1st process in elist
    • mask proposed-ok: processes that proposed the value decided
First phase

• The protocol terminates in the first phase if there are no faults or delays

• The sender:
  - sends a data message (DAT)
  - give the recipients a reliable hash of the message sent using the TTCB Agreement Service

• The TTCB Agreement Service acknowledges the processes that proposed the right hash
  - if all proposed the protocol terminates
BRM-M Sender and Recipient protocol
1 // ——— Phase 1 ———
2 if I am the sender then // SENDER process
3 \[ M := (\text{DAT, my-eid, eist, TTCB-getTimestamp() + } T_1, \text{data}); \]
4 multicast \( M \) to eist except sender; \( n\)-sends := 1;
5 else // RECIPIENT processes
6 read_blocking(M); \( n\)-sends := 0;
7 propose := TTCB-propose(M.eist, M.tstart, TTCB_TBA_RMUTICAST, H(M));
8 do decide := TTCB_decide(propose.tag);
9 while (decide.error \neq \text{TTCB_TBA_ENDED});
10 if (decide.proposed-ok contains all recipients) then deliver \( M \); return;
11 // ——— Phase 2 ———
12 \( M\)-deliver := \( \perp \);
13 mac-vector := calculate macs of (\text{ACK, my-eid, M.eist, M.tstart, decide.value});
14 \( M\)-ack := (\text{ACK, my-eid, M.eist, M.tstart, mac-vector});
15 \( n\)-acks := 0; ack-set := eists in decide.proposed-ok;
16 t-resend := TTCB-getTimestamp();
17 do
18 if (M.type = \text{DAT}) and (H(M) = decide.value) then
19 \( M\)-deliver := \( M \);
20 ack-set := ack-set \cup \{\text{my-eid}\};
21 if (my-eid \notin \text{decide.proposed-ok}) and (\( n\)-acks < \( Od+1 \)) then
22 multicast \( M\)-ack to eist except my-eid; \( n\)-acks := \( n\)-acks + 1;
23 else if (M.type = \text{ACK}) and (M.mac-vector[my-eid] is ok) then
24 ack-set := ack-set \cup \{M.sender\};
25 if (\( M\)-deliver \neq \( \perp \)) and (TTCB-getTimestamp() \geq t\text{-resend}) then
26 multicast \( M\)-deliver to eist except (sender and eists in ack-set);
27 t-resend := t-resend + Tresend; \( n\)-sends := \( n\)-sends + 1;
28 read_non_blocking(M); // sets \( M = \perp \) if no messages to be read
29 while (ack-set does not contain all recipients) and (\( n\)-sends < \( Od+1 \));
30 deliver(M-deliver);

Figure 2. BRM-M protocol.
Example: best case (1st phase only)

- Propose
- DAT msg
- Decide
- H(M)
- H(M), all proposed ok
- Od = k

Diagram shows the sequence of events with phases and messages.
Second phase (II)

- Each process that has the message for which $H(M) =$ value returned by the TTCB Agreement, resends $M$ until:
  - All processes acknowledged:
    - Proposing on time for the TTCB Agreement; or
    - With an ACK
  - Or until it sent $O_d+1$ times:
    - Processes that do not receive are failed
Example: malicious sender

TTCB

P1

P2

P3

P4

TTCB agreement

propose

decide

DAT msg

ACK msg

msg delivery

H(M)

H(M')

H(M)

tstart

Od = 1

© 2008 Paulo Veríssimo
Example: message losses/delays

```
H(M)
H(M)
H(M)

P1
P2
P3
P4

TTCB

TTCB agreement

propose
decide

DAT msg
ACK msg

msg delivery
msg lost
```

Od = 1
Achievements

- Reliable multicast with Byzantine faults requires:
  - asynchronous system: $n \geq 3f+1$ [Bracha&Toueg]
  - synchronous system: no limit ($n \geq f+2$) [Lamport et al.]
- We follow a wormhole-aware model:
  - payload is asynchronous and byzantine-on-failure
  - TTCB is synchronous and crash-on-failure
- We achieve:
  - $n \geq f+2$ without asymmetric crypto (signatures)
  - Efficiency: few phases, high performance
State machine replication on atomic multicast

System architecture

SERVERS

Os1
Local TTB
Host 1

Os2
Local TTB
Host 2

... s_n ...
Local TTB
Host n

TTCB Control Channel

Payload Network

TTTB

Only servers have wormholes

(possibly many) CLIENTS
SMR definition

- Servers are state machines:
  - state variables, commands
- Basic idea: to make all servers follow the same sequence of states, i.e., enforce:
  - Initial state: all servers start in the same state
  - Agreement: all servers execute the same commands
  - Total order: all servers execute the commands in the same order
  - Determinism: the same command executed in the same initial state generates the same final state
Main Contribution

• There is a maximum number $f$ of servers that can be faulty for the system to remain correct.

• With an **homogeneous system model** (asynchronous Byzantine):
  - Minimum: $N=3f+1$ servers
  - 4 servers to tolerate 1 faulty, 7 to tolerate 2 faulty,…

• With a **hybrid system model** (secure wormhole in servers; not in clients):
  - Minimum: $N=2f+1$ servers
  - 3 to tolerate 1 faulty, 5 to tolerate 2 faulty,…
  - This reduction has a huge impact on the system cost: hw, sw, admin (diversity)
Trusted Ordering Wormhole

• The TOW is a wormhole that serves specifically to implement a 2f+1 I-T atomic multicast

• Provides a single service with two purposes:
  - Says when a message can be delivered (which is when f+1 servers have it)
  - Says the order in which it must be delivered

• API:
  - TOW_sent - “I sent a message”
  - TOW_received - “I received a message”

• Output:
  - TOW_decide - “You can deliver the message, order is n”
2f+1 Atomic multicast w/TOW

N=3  f=1

H(M) – a collision-resistant hash function

works the same way with more messages

f+1 servers have M1
order = 1

message delivery
Achievements

• First SMA service for practical byzantine distributed systems with resilience $f$ out of $2f+1$
  - Lower number of replicas reduces cost of hardware + cost of designing different replicas (for fault independence)

• Low time complexity

• Good performance since it does not resort to public key cryptography
Example
Resilient Systems
MAFTIA - Malicious and Accidental Fault Tolerance for Internet Applications

Computer systems can fail for many reasons.

MAFTIA is investigating ways of making computer systems more dependable in the presence of both accidental and malicious faults.
Objectives

• Architectural framework and conceptual model
• Mechanisms and protocols:
  - dependable middleware
  - large scale intrusion detection systems
  - dependable trusted third parties
  - distributed authorisation mechanisms
• Validation and assessment techniques
• Partners
  - DERA/Qinetiq, Malvern (UK) - Tom McCutcheon / Sadie Creese
  - IBM, Zurich (CH) - Marc Dacier / Michael Waidner
  - LAAS-CNRS, Toulouse (F) - Y. Deswarte / D. Powell
  - Newcastle University (UK)(Coord.) - R. Stroud / Brian Randell
  - Universität des Saarlandes (D) - Michael Steiner
  - Universidade de Lisboa (P) - Paulo Veríssimo / Nuno F. Neves
• EU coordinator - Andrea Servida

http://www.research.ec.org/maftia
Architecture Overview
Host architecture

Hardware
Untrusted Hardware
Trusted Hardware

Local Support
Runtime Environment (JVM+ Appia)
O.S.
TTCB
Security Kernels

Distributed Software
Applications
Activity Support Services
Communication Support Services
Multipoint Network

AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service
trusted— vs. untrusted— hardware

Most of MAFTIA’s hardware is untrusted, but small parts considered trusted in the sense of tamperproof by construction.
Architecture Overview
Host architecture

- Security kernels materialising fail-controlled subsystems
  - Trusted to execute a few functions correctly, albeit immersed in an environment subjected to malicious faults
  - Local security kernels (Java Card)
  - Distributed security kernels (Trusted Timely Computing Base)

AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service
**Trusted Timely Computing Base DSK**

- TTCB is a distributed security kernel that provides a minimal set of trusted and timely services
- Construction principles: interposition, shielding, validation
- Classic Trusted Computing Base aims at fault prevention, while the TTCB aims at fault tolerance
- TTCB can be a: special hardware module (e.g. tamperproof device); secure real-time microkernel running on a workstation or PC underneath the OS
- TTCB control channel has to be both timely and secure: virtual network with predictable characteristics coexisting with the payload channel; separate physical network
Architecture Overview
Host architecture

- Run-time environment extending OS capabilities
- Hiding heterogeneity by offering a homogeneous API and framework for protocol composition

AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service
Architecture Overview
Host architecture

- modular and multi-layered middleware
- neat separation between different functional blocks

AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service
Modular Group Architecture

- **Multipoint Network**
  - Multipoint addressing and routing
  - Basic secure channels and envelopes
  - Management Communication prots
  - Appia APIs for mcastIP, Ipsec, SNMP

- **Communication Services**
  - Distributed Cryptography (threshold public key)
  - Group Communication (reliability and order props)
  - Byzantine Agreement
  - Time and Clock Synchronisation

- **Main Activity Services**
  - Replication management
  - Key Management
  - Transactional Management

- **Activity Support Services (AS)**
  - Replication management
  - Key Management
  - Transactional Management

- **Communication Support Services (CS)**
  - Distributed Cryptography (threshold public key)
  - Group Communication (reliability and order props)
  - Byzantine Agreement
  - Time and Clock Synchronisation

- **Runtime Environment**
  - Appia+JVM+OS

- **Membership and Failure Detection**
  - Participant level
  - Site level
  - Physical Network

- **Applications**
  - Participant level
Group Communication on Asynchronous model

- Stack of protocols for (among other applications) intrusion-tolerant replicated servers on an asynchronous wide-area setting
- Main characteristics of the model: asynchronous; static and open groups; up to n/3 corrupted processes (f < n/3); threshold crypto; manual and trusted key distribution

<table>
<thead>
<tr>
<th>Secure Causal Atomic Broadcast</th>
<th>provides confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Broadcast</td>
<td>ordering of client requests</td>
</tr>
<tr>
<td>Multi-value Agreement</td>
<td>agreement on request values</td>
</tr>
<tr>
<td>Broadcast Primitives</td>
<td>reliable and consistent broadcasts, agreement on Y/N questions</td>
</tr>
<tr>
<td>Binary Agreement</td>
<td>threshold cryptosystem and digital signature scheme</td>
</tr>
<tr>
<td>Threshold Cryptography</td>
<td></td>
</tr>
<tr>
<td>Reliable Point-to-Point Links</td>
<td></td>
</tr>
</tbody>
</table>
Byzantine-Reliable Multicast on Timed Model with TTCB

Byzantine Reliable Multicast Protocol (1 Phase)

5-Node Delivery Times

BRM/mcast average delivery times (milliseconds)
### High-performance RITAS randomized protocol stack

- **Asynchronous**: no timing
- **Byzantine faults**: No more than $f=\left\lfloor \frac{n-1}{3} \right\rfloor$ corrupt processes out of $n$
- **Every pair of processes share a symmetric key**

<table>
<thead>
<tr>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Consensus</td>
</tr>
<tr>
<td>Atomic Broadcast</td>
</tr>
<tr>
<td>Multi-valued Consensus</td>
</tr>
<tr>
<td>Binary Consensus</td>
</tr>
<tr>
<td>Reliable Broadcast</td>
</tr>
<tr>
<td>Echo Broadcast</td>
</tr>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>IPSec AH</td>
</tr>
</tbody>
</table>

#### Protocols Implemented in RITAS

#### Standard Internet Protocols
IT Transactions with Error Masking

- A CORBA-style transaction service, standard ACID properties
- Support for multiparty transactions
- Uses error masking to tolerate intrusions
- Application of hybrid failure assumptions
IT Authorisation Service

Authorization Server

Security kernel

JavaCard

u

Security kernel

fs2

Security kernel

ps1

p4

f3
• finding solutions to the problems of the high rate of false positive and false negative alarms generated by existing solutions
• these false alarms can also be due to attacks against the IDS itself, therefore the need to design an IDS which is itself tolerant to intrusions
• study and evaluate how notions such as fault injection, diversity and distributed reasoning can address the weaknesses of existing solutions
Verifying MAFTIA protocols

Abstract

Concrete

Formal methods (e.g., CSP)

Cryptography

Faithful abstraction
OASIS

ORGANICALLY ASSURED & SURVIVABLE INFORMATION SYSTEMS
ORGANICALLY ASSURED & SURVIVABLE INFORMATION SYSTEMS
Dr. Jaynarayan Lala – jlala@darpa.mil, 703-696-7441

Organically Assured Survivable Information Systems, OASIS Demonstration and Validation Program

Some Attacks Will Succeed

3rd Generation
(Operate Through Attacks)

Intrusion Tolerance
Graceful Degradation

Big Board View of Attacks
Real-Time Situation Awareness & Response

Hardened Core
Performance
Functionality
Security
**Intrusion Tolerant Architecture**

**Objectives**
- Construct intrusion-tolerant architectures from potentially vulnerable components
- Characterize cost-benefits of intrusion tolerance mechanisms
- Develop assessment and validation methodologies to evaluate intrusion tolerance mechanisms

**Technical Approach**
- **Real-Time Execution Monitors:** In-line reference monitors, wrappers, sandboxing, binary insertion in legacy code, proof carrying code, secure mobile protocols
- **Error Detection & Tolerance Triggers:** Time and Value Domain Checks, Comparison and Voting, Rear Guards
- **Error Compensation, Response and Recovery:** Hardware and Software Redundancy, Rollback and Roll-Forward Recovery
- **Intrusion Tolerant Architectures:** Design Diversity, Randomness, Uncertainty, Agility
- **Assessment & Validation:** Peer Review Teams, Red Team, Assurance Case (Fault Tree, Hazard Analysis, Formal Proofs, Analytical Models, Empirical Evidence)

**Schedule**

<table>
<thead>
<tr>
<th>Phase</th>
<th>1/99</th>
<th>1/00</th>
<th>1/01</th>
<th>1/02</th>
<th>1/03</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-time Execution Monitors, Error Detection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developing Technology Drops</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase II</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Compensation, Response, Recovery</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Protected**

**Users/Clients**

**COTS Servers**
Bibliography

Books:

Book chapters: