Travelling through Wormholes:
a new look at Distributed Systems Models

Paulo Esteves Veríssimo

Navigators Group,
LaSIGe, Laboratory for Large-Scale Informatic Systems
Univ. Lisboa
pjv@di.fc.ul.pt
http://www.di.fc.ul.pt/~pjv
Problem Motivation

• Design and deployment of distributed applications is faced with the confluence of antagonistic aims:
  - between what is required by applications, and what is given by the supporting infrastructure/environment

• Current and future large, massive-scale pervasive and/or ubiquitous computing systems will amplify this:
  - very high numbers of players, very large distances, geographical scope, topology and interconnections no longer a given, ill-defined COTS component properties

• Key lies with a changing notion of service guarantees:
  - on what have always been the fundamental issues, e.g., consistency, synchronism, reliability, availability, predictability, security, ...
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

This was the insight that led to the **Wormholes hybrid distributed system model**
Take time/synchrony facet

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

- discrete
- exp/enforcing
- event/perpet

- continuous
- expecting
- eventual

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

• **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)
Homogeneous distr. sys. models

- Ps
- Pr
- Pt
- Pu
- PV
- WG

synchronous
asynchronous
Homogeneous distr. sys. models

synchronous

asynchronous
Shortcuts vs. detours

- We propose to render the solution simpler, without “making” the problem easier!
- The insight:
  - uncertainty is not ubiquitous or everlasting
  - proactively achieve predictability
  - tolerate uncertainty
- The workhorses:
  - Wormholes model
  - Architectural hybridization

- **Uncertainty and Predictability: Can they be reconciled?**
- **Travelling through Wormholes: a new look at Distributed Systems Models**
  Paulo Veríssimo. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory), vol. 37, no. 1, (Whole Number 138), 2006.
any-synchrony System w  any-synchrony System p
Wormholes

- New design philosophy for architecting and programming distributed systems:
  - constructs with privileged properties that endow systems with the capability of evading the uncertainty or weakness of the environment ("taking a shortcut") for certain crucial steps of their operation, in order to achieve overall strong properties otherwise impossible or complex or expensive.

- Architectural hybridization:
  - loci with different properties
  - "strong" and "weak" components
  - e.g., trustworthy components
Example of deployment of systems with wormholes
Example of deployment of systems with wormholes
Example of deployment of systems with wormholes
Taking shortcuts i.s.o. detours

- **OBJECTIVE:**
  - solve most timed or non-timed problems with highest possible coverage

- overcome algorithmic hardness (e.g., w.r.t. asynchronism, maliciousness, etc.) through computing models aware of *wormholes*

- enforce hybrid behaviour (“strong” and “weak” components) by *architectural hybridization*

- implement strong q.b. components (*trusted-trustworthy*)
Designing algorithms with wormholes
(aka hybrid distributed systems models)

Assume basic system model, e.g.
- asynchronous and Byzantine failures

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

Design your algorithms and
prove them correct

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

any-synchrony System w
any-synchrony System p
Designing algorithms with wormholes (aka hybrid distributed systems models)

Assumptions substantiated by architectural hybridization

Design 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.
Proof-of-concept systems with wormholes
Proof-of-concept: COTS-based TCB Reference Architecture

Linux Application

--- TCB Specific ---

API

Linux Application

Regular Linux OS

Self-checking Mechanisms

Fail-Silence Switch

RT-Linux

System HW Resources

(RAM, Processor, Interrupts, etc.)

TPM

RT-Linux Driver

Fast-Ethernet Network

Regular Networking Infrastructure

Regular Linux Driver

March, 2004 @ Papeete, Fr Poly
pjv@di.fc.ul.pt

IFIP WG10.4 -- 45th meeting and Workshop
Proof-of-concept systems with wormholes

any-synchrony System w  any-synchrony System p
Proof-of-concept: Distr. crash failure synch. wormhole

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 System w any-synchrony System p
A (necessarily brief) birds-eye view of some results
Some intrusion tolerance algorithmic results

- **Using Wormholes**
  
- **Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model**, SRDS 2002
  - fast and secure BA (8ms)

- **How to Tolerate Half Less One Byzantine Nodes in Practical Distr. Systems**, SRDS 2004
  - Low time complexity, first SMA byzantine f out of 2f+1

- **Low Complexity Byzantine-Resilient Consensus**, Distr. Comp. journal, 2004
  - Low latency (degree 2)

- **Solving Vector Consensus with a Wormhole**, IEEE TPDS journal, 2005
  - Fully async, low latency, low complexity
Recent results on exhaustion safety

- *Securing exhaustion safety*

- *How Resilient are Distributed f Fault/ Intrusion-Tolerant Systems?*
  DSN 2005

- *Resilient State Machine Replication*
  PRDC 2005

- *Proactive Resilience through Architectural Hybridization*
  ACM SAC 2006
Trusted Timely Computing Base (TTCB)

Properties:
- trusted and timely execution assistant; trusted timing failure detector
- secure (can only fail by crashing)
- real-time (capable of timely behavior)
- correct processes can interact securely with the TTCB

Assists the execution of fault-tolerant algorithms:
- provides a trusted environment for crucial steps

Can be built (there is a prototype)

Trusted Timely Computing Base (TTCB) prototype

- **Properties:**
  - trusted and timely execution assistant; trusted timing failure detector
  - secure (can only fail by crashing)
  - real-time (capable of timely behavior)
  - correct processes can interact securely with the TTCB

- **Assists the execution of fault-tolerant algorithms:**
  - provides a trusted environment for crucial steps

- **Can be built (there is a prototype)**


Downloadable from
Navigators group: