A FIT Event Broker for trustworthy infrastructure monitoring and management

António Casimiro
University of Lisbon Faculty of Sciences
LASIGE – Navigators group

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

61st IFIP WG 10.4 Meeting, January, 2012

Outline

- Motivation: the TRONE project
- Goals and challenges
- FIT broker architecture and operation
- Other work in TRONE
- Conclusions

The TRONE project

- Develop innovative solutions for Network Operation, Administration and Management
  - **Proactive** hazard reduction: architectural robustness
  - **Reactive** hazard reduction: detection and recovery

- Achieve trustworthy network operation
  - Solutions for dynamic dependability & security enforcement
  - Deal with increasing levels of accidental and malicious faults
    - Diagnosis, detection
    - **Prevention/tolerance**
    - Automatic reconfiguration
  - Provide architectural solutions and resilient components

Focus of this presentation

Why we need TRONE?

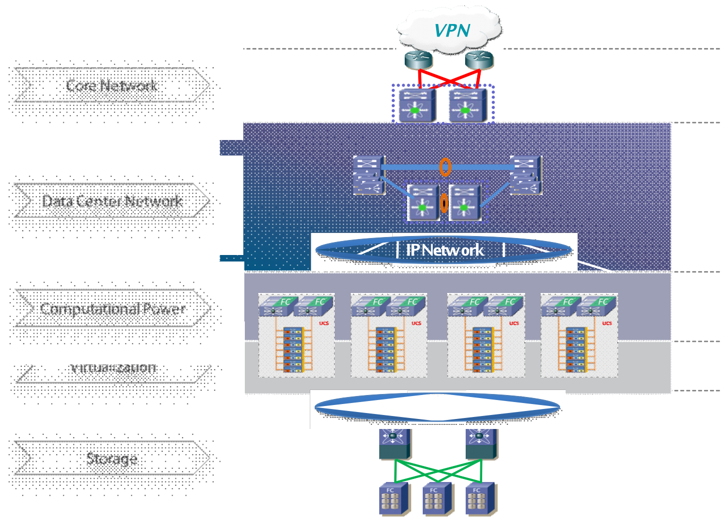
Trustworthy and Resilient Operations in a Network Environment

- **Technology push**:
  - Next Generation Networks, Cloud Computing
  - Need for seamless integration of new and heterogeneous technologies
- **Consumer pull**:
  - More demanding requirements
  - Increased QoS and **QoP** (fast is not enough!)
- The confluence of these forces leads to:
  - Increased operational risks
  - Inadequate network operation and management

Example scenario:

Portugal Telecom Cloud Computing Infrastructure

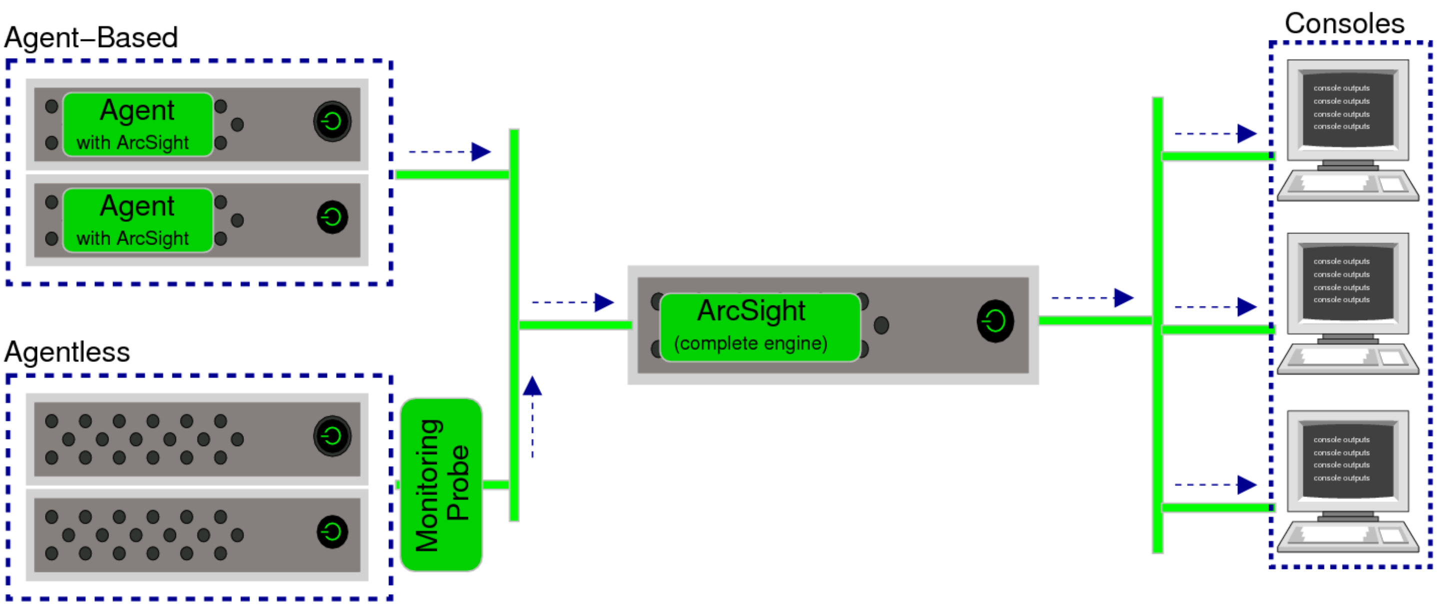
- Cloud computing environments:
  - Will be, in the next few years, the **hottest topic** in Portugal Telecom services portfolio
  - Present a set of new **challenges regarding security controls**
  - Same infrastructure is used by clients with strong security awareness and others that do not share this awareness
  - The reach and impact of an attack is potentially greater than in traditional IT infrastructures



Portugal Telecom Reference Architecture

Example scenario:
Portugal Telecom Cloud Computing Infrastructure

- Focusing on a specific problem:
  - Centralized monitoring approach



Goals and challenges

- Overarching goals:
  - To provide support for trustworthy and resilient monitoring of cloud/datacenter infrastructures
  - To achieve improved Quality of Protection while considering Quality of Service (performance) needs

- Some specific challenges:
  - Deal with large flows of information (events)
  - Support different kinds of events (e.g. criticality)
  - Low intrusiveness and easy integration

Fault and Intrusion Tolerant (FIT) Event Broker

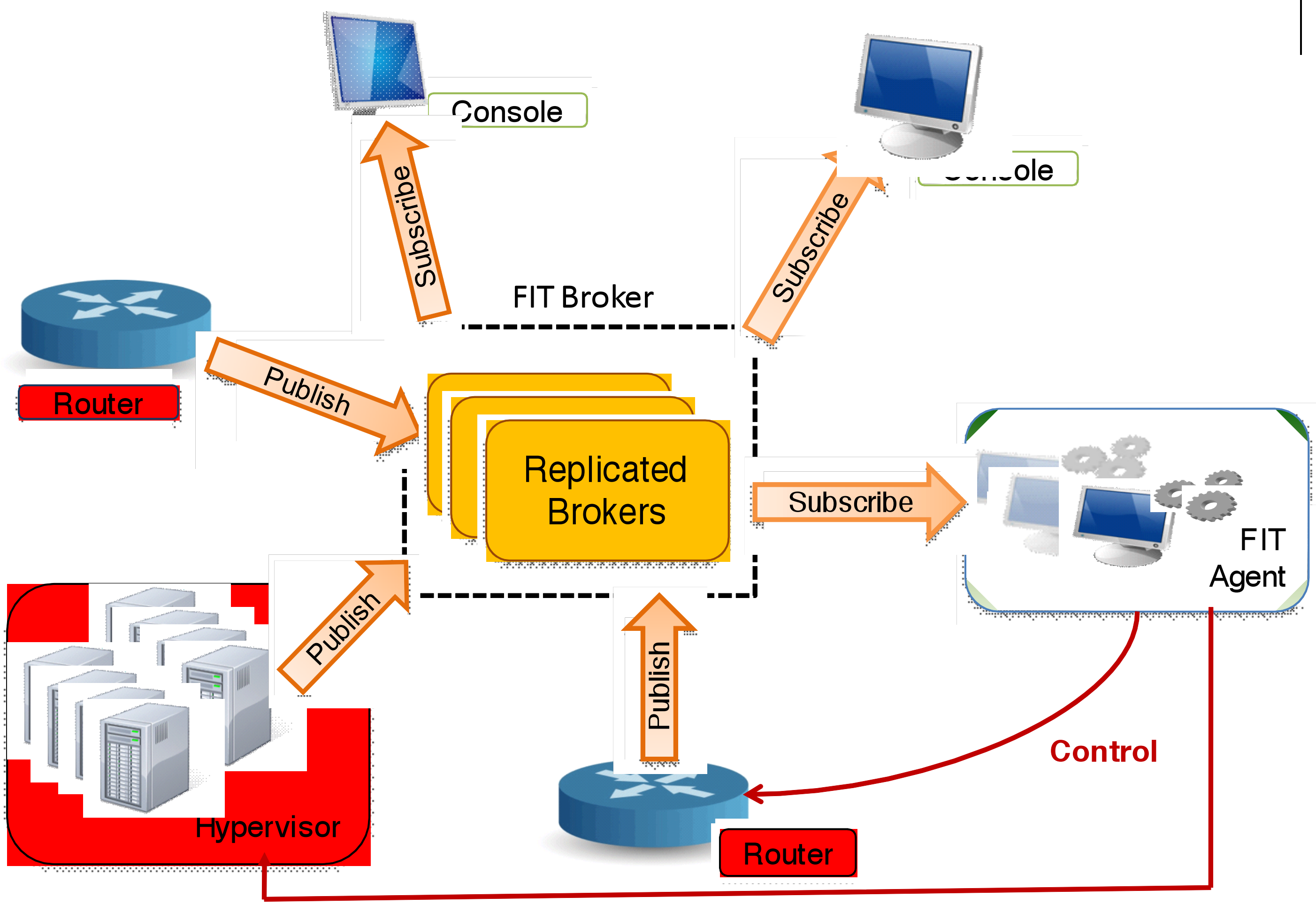
Assumptions

- System entities:
  - Probes, event collectors/brokers, consoles
  - Some event processing may be done by collectors
- Fully connected network
  - E.g., all the entities lie in the same monitoring VLAN
- Partially synchronous system
  - Clocks may be used to timestamp events
- Faults
  - Some FIT brokers may fail in a Byzantine way (e.g. be attacked)
  - We do not require/enforce clients (probes/consoles) to be correct
    - If this is a problem for monitoring, then it must also be solved

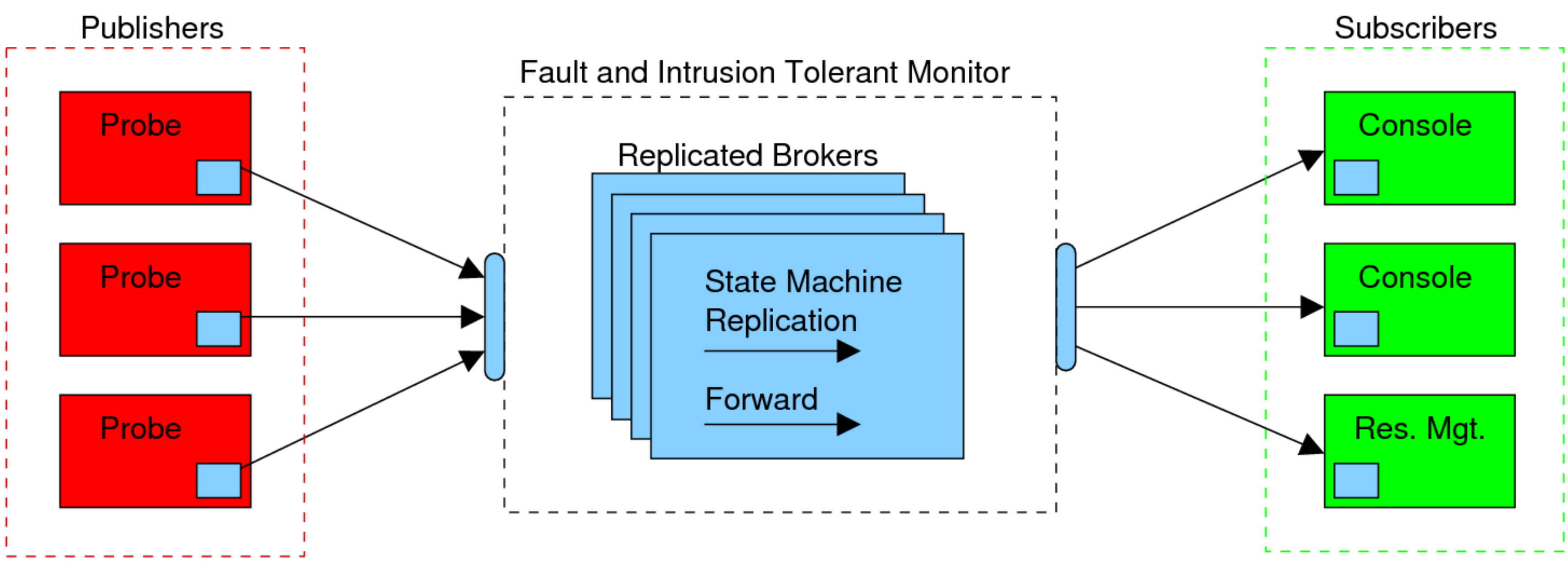
Baseline design options

- Topic-based Publish-Subscribe paradigm
  - Good fit to considered scenarios
- State Machine Replication
  - Active replication is better for Byzantine fault tolerance
  - *f* out of *n* replicas of a FIT Broker may fail in a Byzantine way
- Public-key cryptography
  - Client authentication, avoid attacks from malicious probes
- Event channels with support for QoP and QoS
  - Differentiated event handling, on a channel basis
  - Differentiated fault-tolerance support (e.g. crash only or BFT)
  - Possible support for ordering, urgency and other requirements

FIT Monitoring system:
Overview



FIT Monitoring system:
High level architectural view



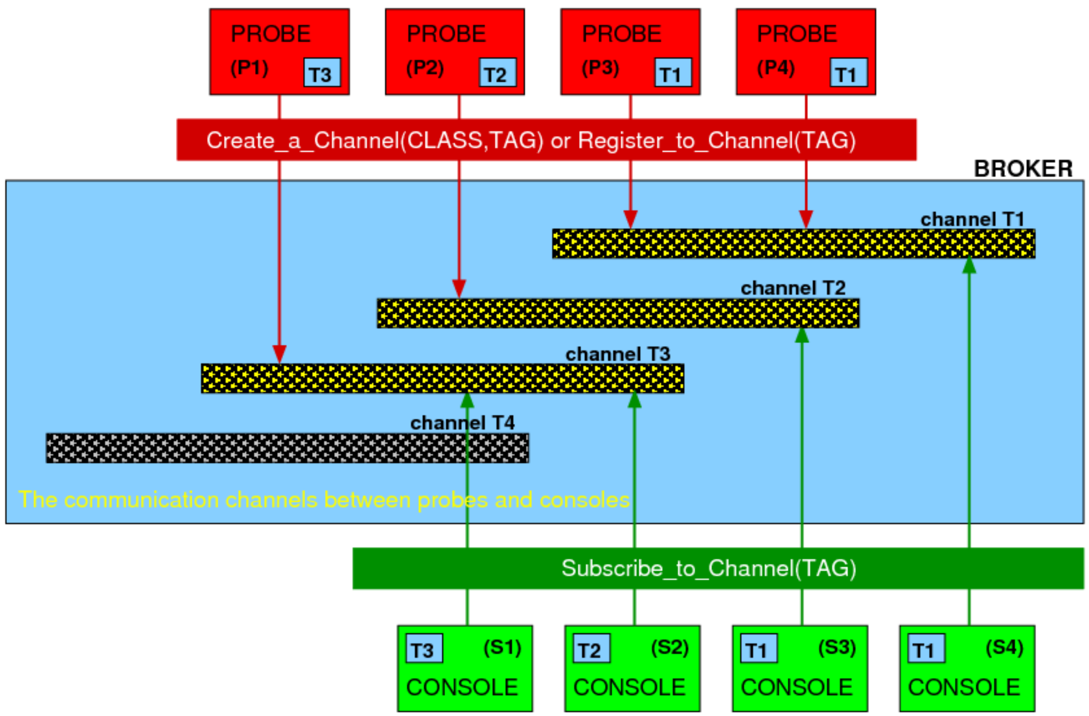
FIT Event Broker
Basic concepts

- Event Channels
  - Fundamental abstraction to differentiate event flows within the event broker
  - An event channel is identified (within the system) by a TAG
  - The characteristics of event channels are set by means of a CLASS attribute
  - Several event channels may be created, defining communication domains
  - An event channel may be used to transmit specific kinds of events
    - For instance: network, storage, security threats, …
- CLASS
  - Defines the desired QoS and QoP for an event channel
    - Fault-tolerance (e.g. whether the event channel/service should be tolerant to crash faults or to Byzantine faults)
    - Ordering (e.g. whether the events transmitted through the event channel should be delivered in the same order to all subscribers, or any order is acceptable)
    - Priority (e.g. whether the event flow is allocated more bandwidth within the broker, or events may be processed before events from other channels within the broker)
  - Several channels with the same CLASS may coexist

FIT Event Broker

Interface

- *Create event channel*
  - In: TAG and CLASS
- *Register to channel*
  - In: TAG
- *Publish event*
  - In: EVENT
- *Subscribe to channel*
  - In: TAG
- *Receive event*
  - Out: EVENT
- *Destroy event channel*
  - In: TAG

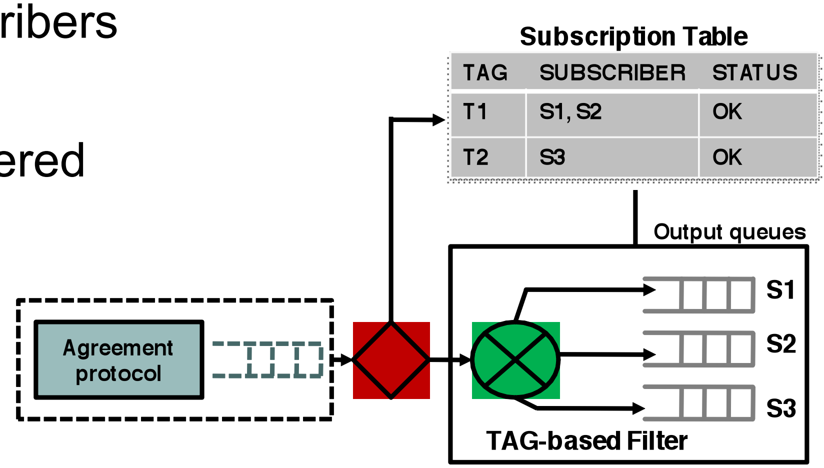


FIT Event Broker

Internal state

- From the SMR perspective, it is important to identify the relevant state that needs to be maintained consistent across replicas
  - Data related to the broker configuration
    - Existing channels and their CLASS
    - Registered publishers and subscribers
  - Data related to events
    - Events that are ready to be delivered

- All client input that affects the state of the FIT broker state (e.g. channel and subscription data, some events) must be handled as a state machine command



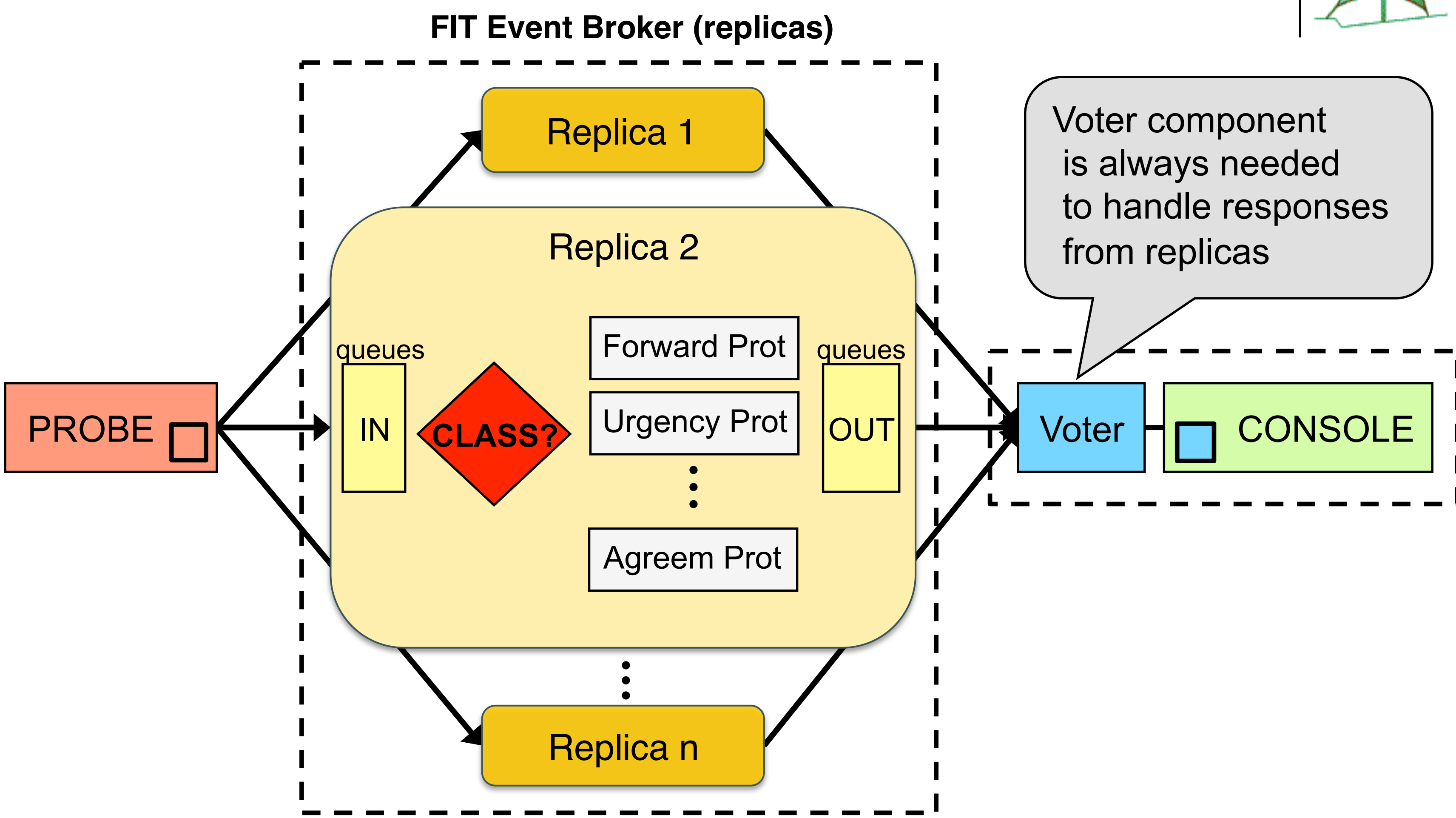
FIT Event Broker

Operation

- Depending on requirements (determined by the channel CLASS), input events are handled by different protocols within the FIT broker
- Crash-resilient channel, no order requirements
  - No consistency among replicas is needed
  - Simple forwarding protocols
  - High performance – adequate to most periodic and non-critical monitoring events
- Byzantine-resilient channel
  - Agreement is needed
  - Performance implications – adequate to critical monitoring events

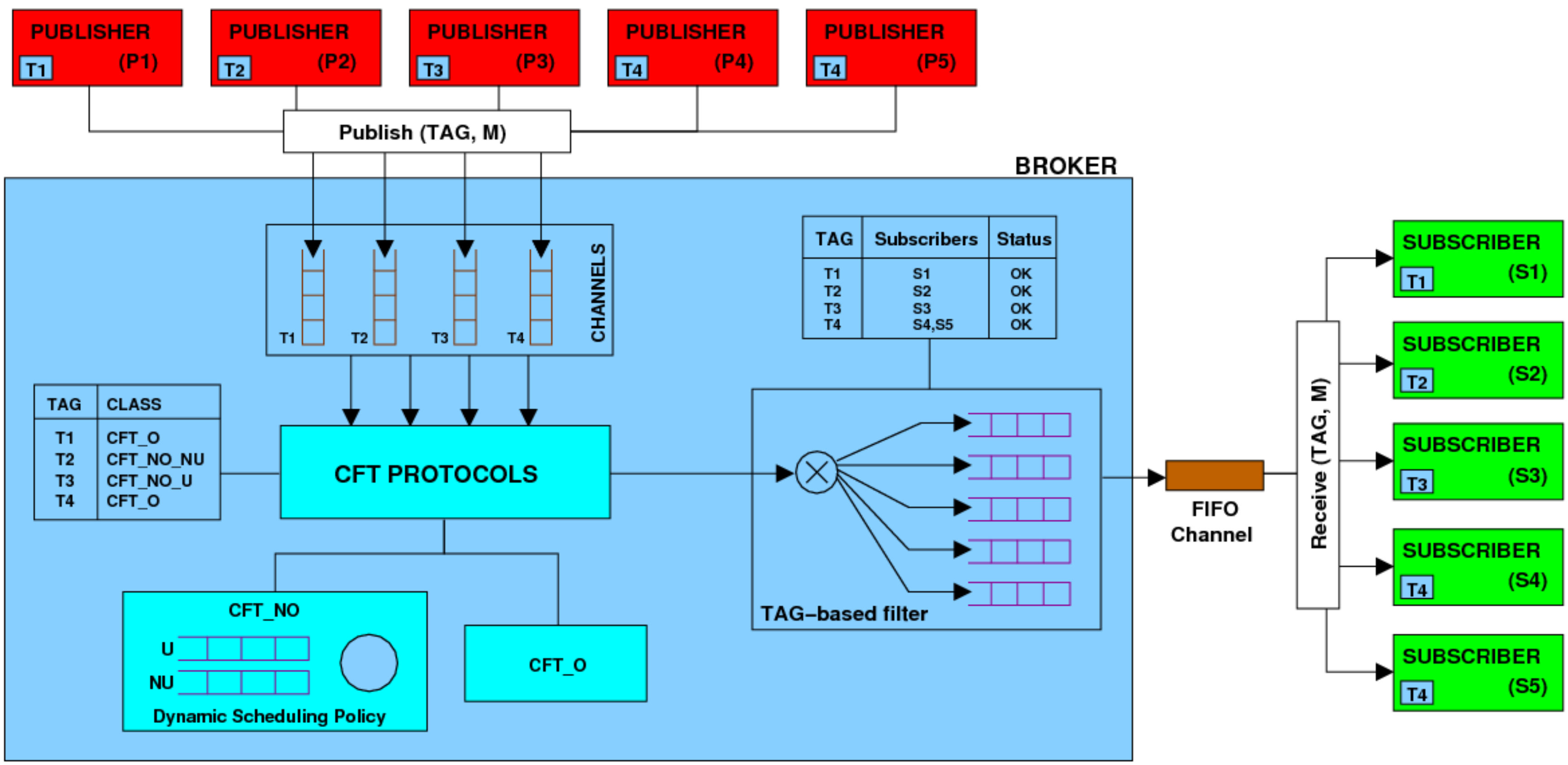
FIT Event Broker

Internal event processing



FIT Event Broker

Crash-tolerant instantiation

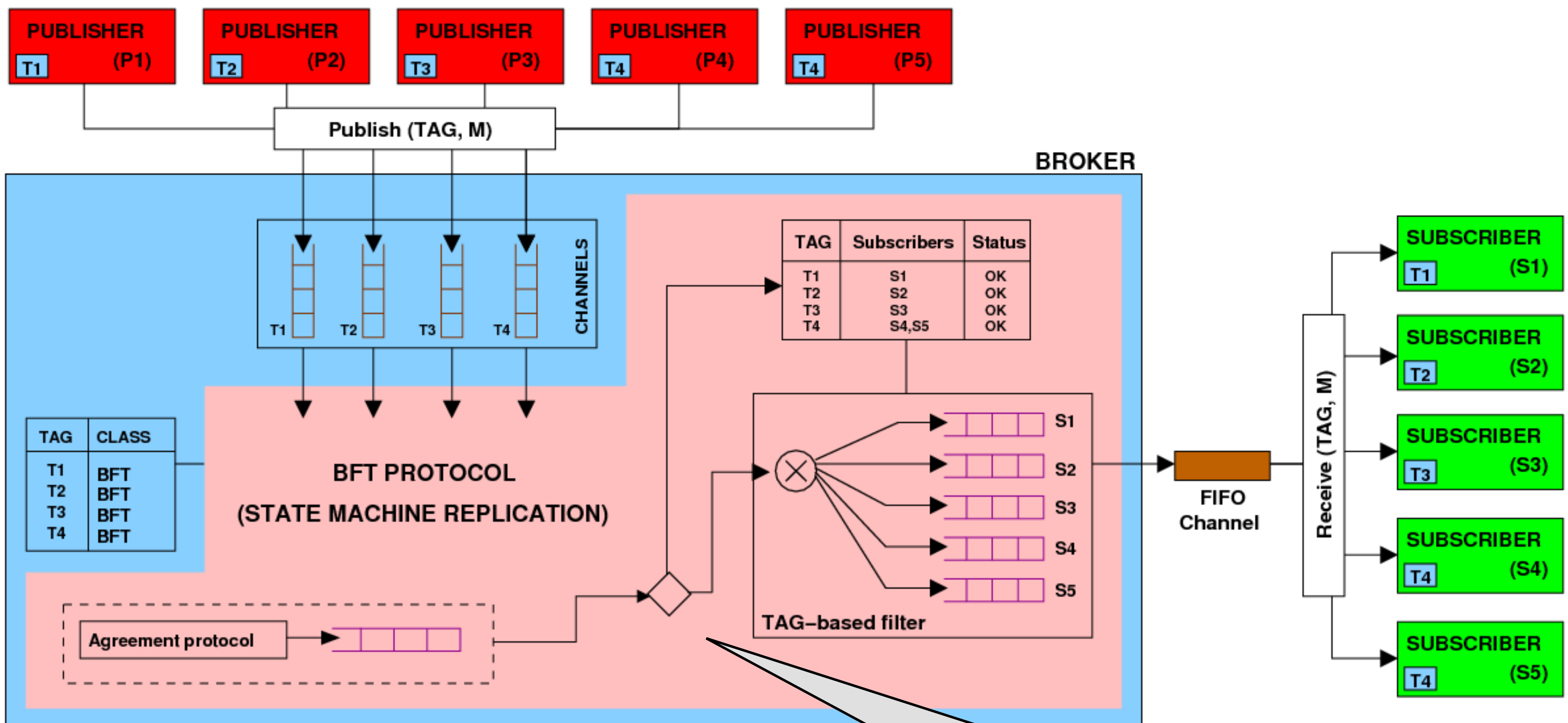


FIT Event Broker

BFT instantiation

FACULDADE DE CIÊNCIAS UNIVERSIDADE DE LISBOA

Navigators



We use **BFT-SMaRt** as a fundamental building block in the implementation of the FIT event broker

BFT-SMaRt

- Java-based platform for BFT SMR, available at ***http://code.google.com/p/bft-smart/***
- Actively being developed and improved in our group
- BFT SMR “common” features
  - State machine programming model
  - $n \geq 3f+1$ replicas required
  - Many bugs (but not as many as competitors) ☺
- Advanced features
  - Replica recovery (state transfer)
  - Reconfigurations
  - Extensible API: e.g. custom voter

BFT Replication Library

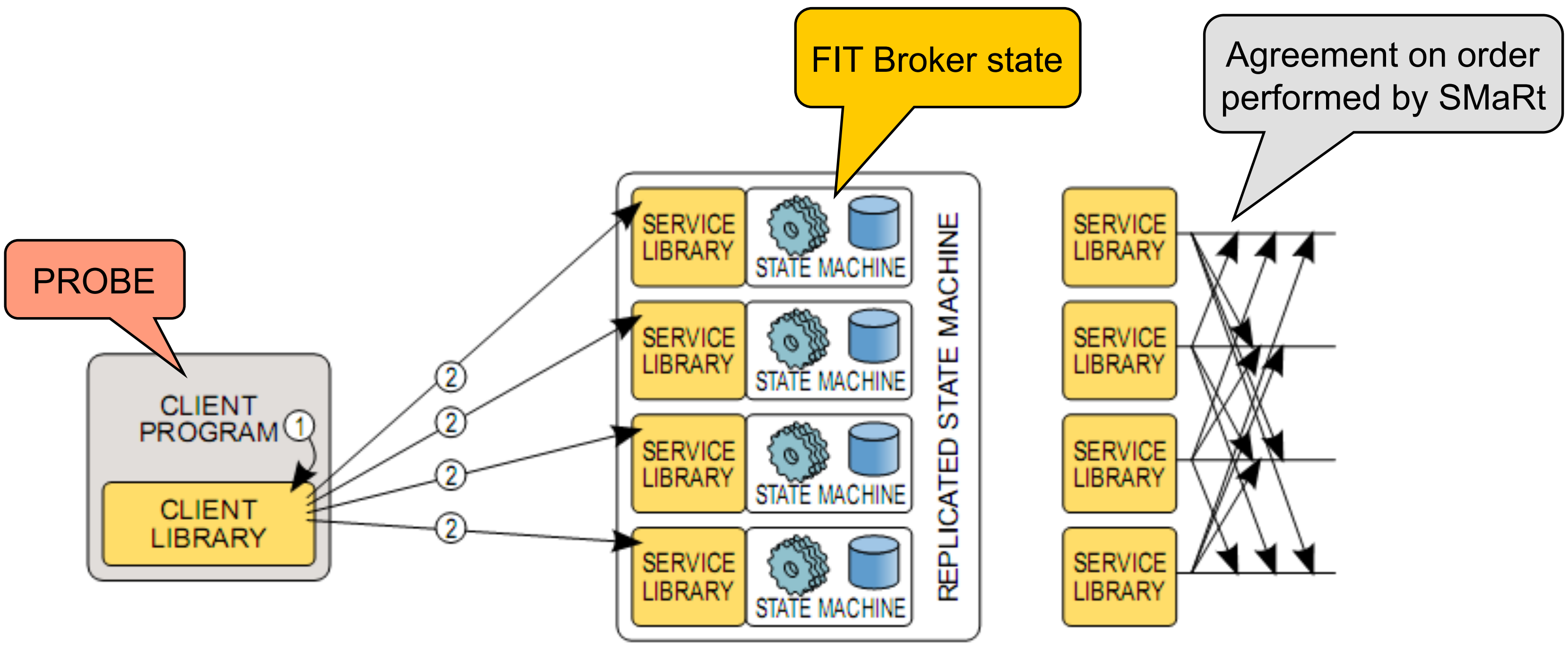
Programming interface

- A very simple and constrained API for implementing state machine replication

```
//Client API
public class ServiceProxy ... {
  ...
  public byte[] invoke(byte[] command, boolean readOnly){
  ...
}

//Server API
public abstract class ServiceReplica ... {
  ...
  public abstract byte[] executeCommand(int clientId,
         long timestamp, byte[] nonces, byte[] command);
  public abstract byte[] getState();
  public abstract void setState(byte[] state);
}
```

Using SMaRt
Service invocation

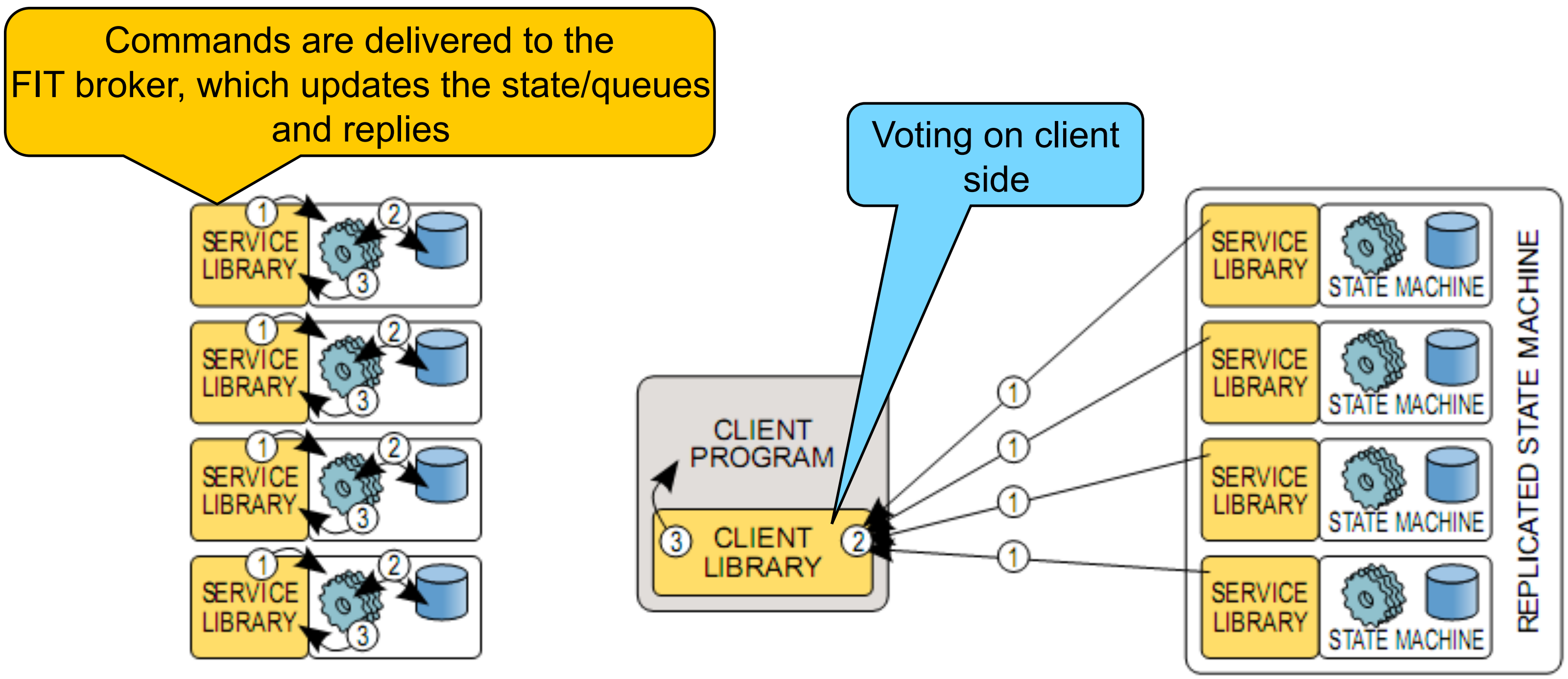


(a) client program (1) calls library library (2) sends call to replicas

(b) service libraries agree on message order

Using SMaRt

Execution and response



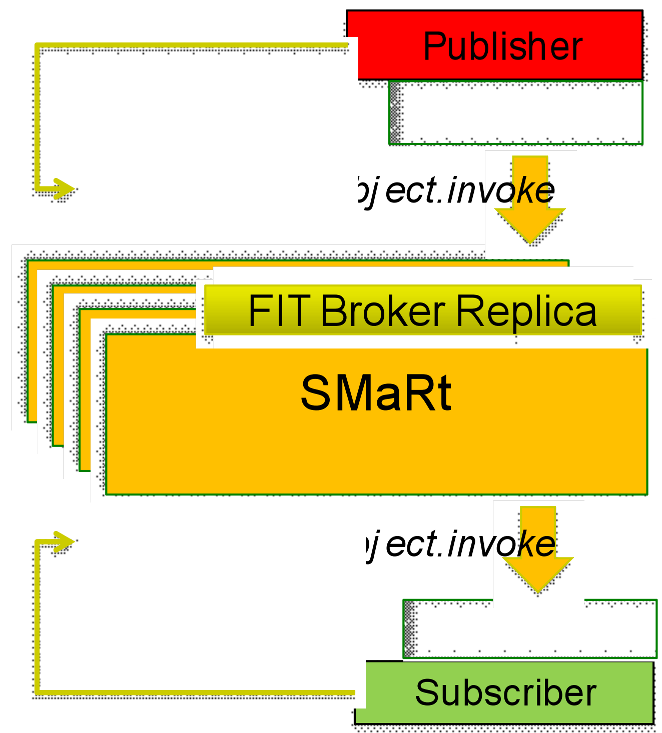
(c) library (1) calls state machine
state machine (2) can read and modify state
state machine (3) replies to library

(d) service library (1) replies to client
client library (2) collects replies and vote
library (3) replies to client program

Implementation & Evaluation

- The FIT Broker is currently being implemented
- On-going work on SMaRt integration

- Evaluation:
  - Throughput
    - Aim is to deal with 40K events/sec
  - Resilience
    - Measure performance under attack
    - Verify recovery and reconfiguration capabilities



Other work in TRONE

Failure Diagnosis
Overview

- Diagnosing problems
  - Creates major headaches for administrators
  - Worsens as scale and system complexity grows
- Goal: automate failure diagnosis and get proactive
  - Failure detection and prediction
  - Problem determination ("automated fingerpointing")
  - Problem visualization
- How: Instrumentation plus statistical analysis

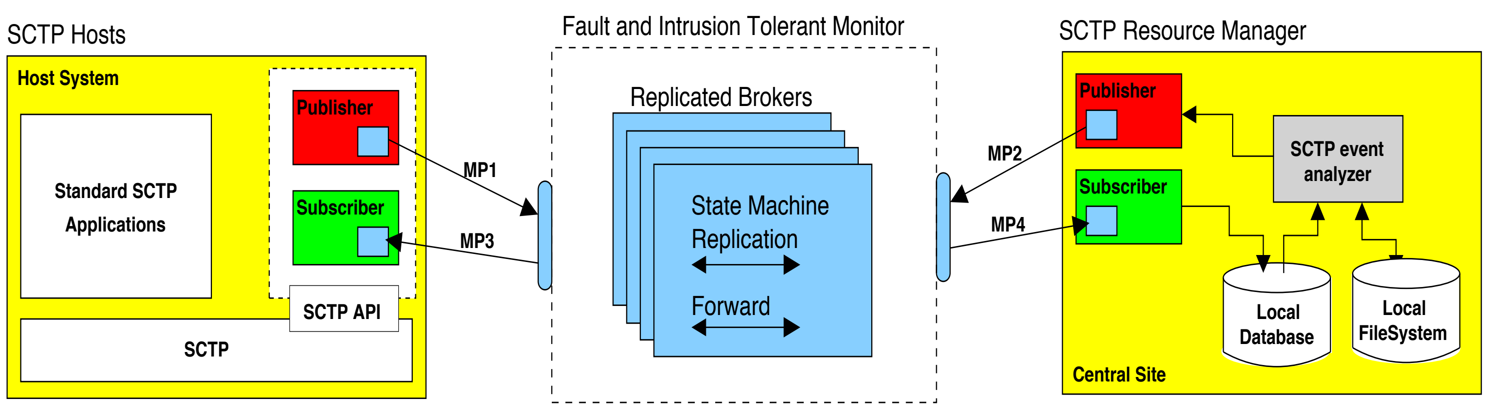
Failure Diagnosis

Goals and Non-Goals

- Goals of the failure diagnosis algorithm
  - Lightweight and Transparent: use metrics that can be collected with minimum overhead and without modifying the applications
  - Scalable: low complexity so that it can scale to several nodes used in the cloud computing infrastructure
  - Versatile: Should work well with all the different kinds of applications that might run on the cloud computing infrastructure
- Non-goals (for now)
  - Tracing problem down to offending line of code
  - Online implementation

Multi-homing

- Objectives:
  - Improved communication resilience
  - Client multi-homing at the transport layer
  - Use of Stream Control Transport Protocol (SCTP)
  - Interaction between SCTP and the FIT Event Broker for trustworthy handling of SCTP monitoring data



Conclusions

- TRONE will contribute to improve the resilience of Portugal Telecom's datacenter monitoring
- Excellent opportunity to
  - Design
  - practically apply
  - and verify the effective benefits of our solutions

Thank you!

Visit us at

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