On the Design of Fault-Tolerance in a Decentralized Software Platform for Power Systems

Purboday Ghosh, Scott Eisele, Abhishek Dubey, Mary Metelko, Istvan Madari, Peter Volgyesi, Gabor Karsai
Institute for Software-Integrated Systems, Vanderbilt University

Supported by DOE ARPA-E under award DE-AR0000666

Outline

- Software for Smart Grid
- RIAPS fundamentals
- Fault management architecture
- Example: Transactive Energy App
- Summary

The Energy Revolution: Big Picture

From centralized to *decentralized* and *distributed* energy systems

Changing Generation Mix

Electric Vehicles

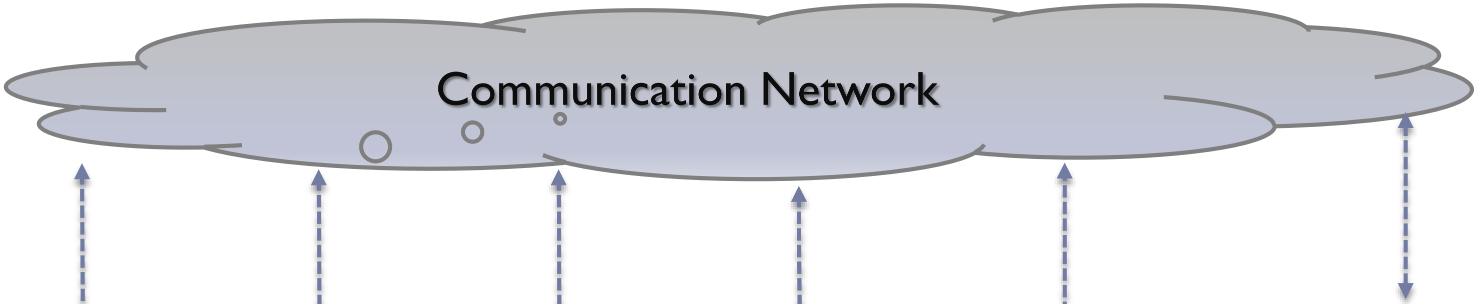
Transactive Energy

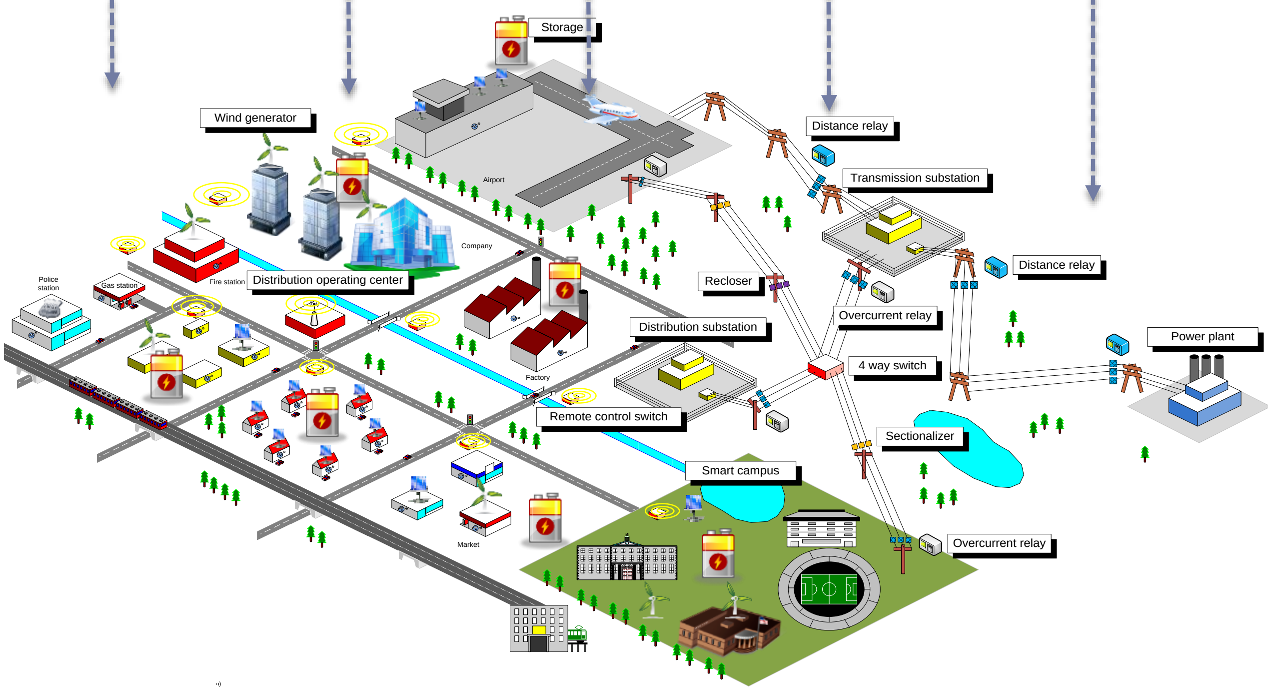
Decentralization

Needs: Distributed 'grid intelligence' *for*

- *Monitoring + control locally and on multiple levels of abstraction*
- *Transactions among peers*
- *Real-time analytics*
- *Autonomous and resilient operation*

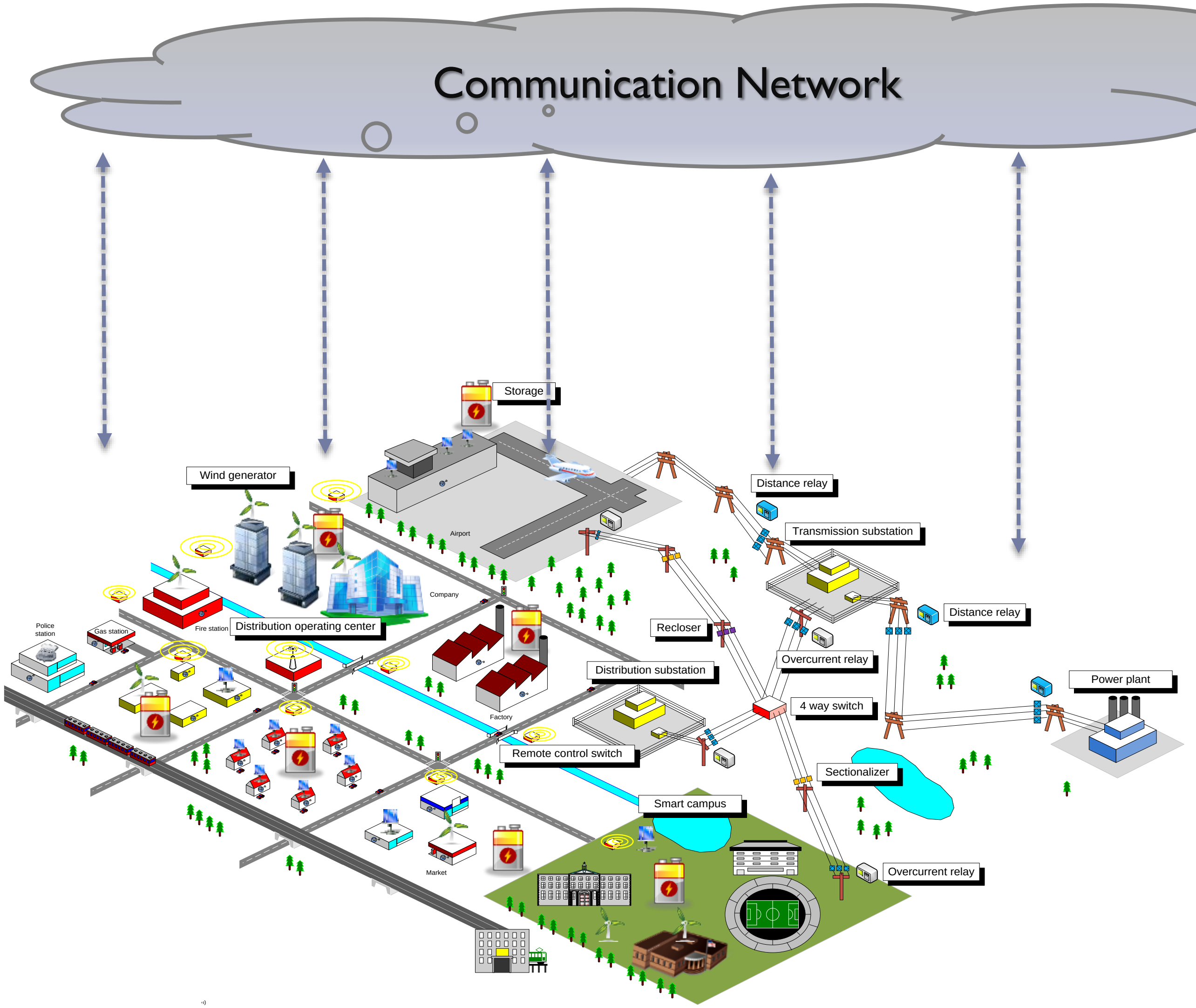
The control picture has not changed

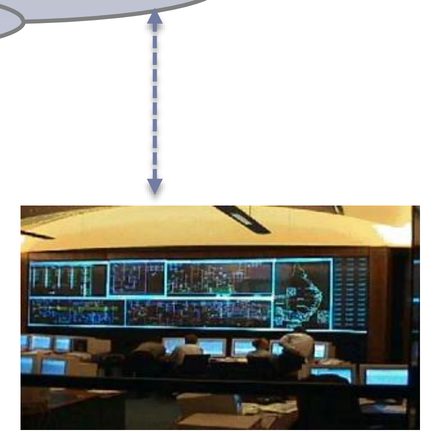




Distribution: Centralized SCADA system managed by the utility company

The control picture has not changed



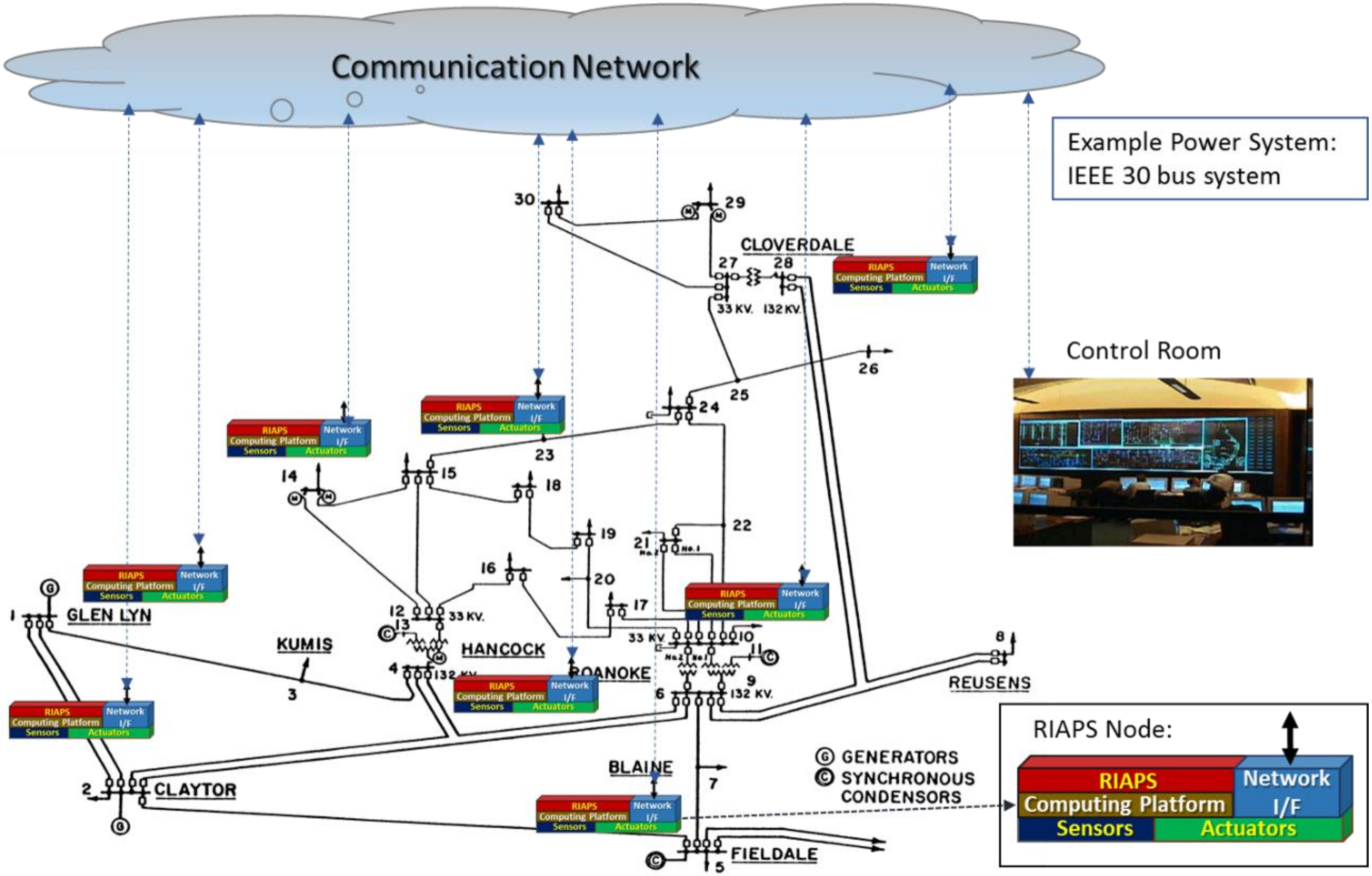


Problems

- Distributed Control
- Network latency
- Lack of interoperability
- Robust/resilient software
- Cyber-security
- Integration challenges
- …

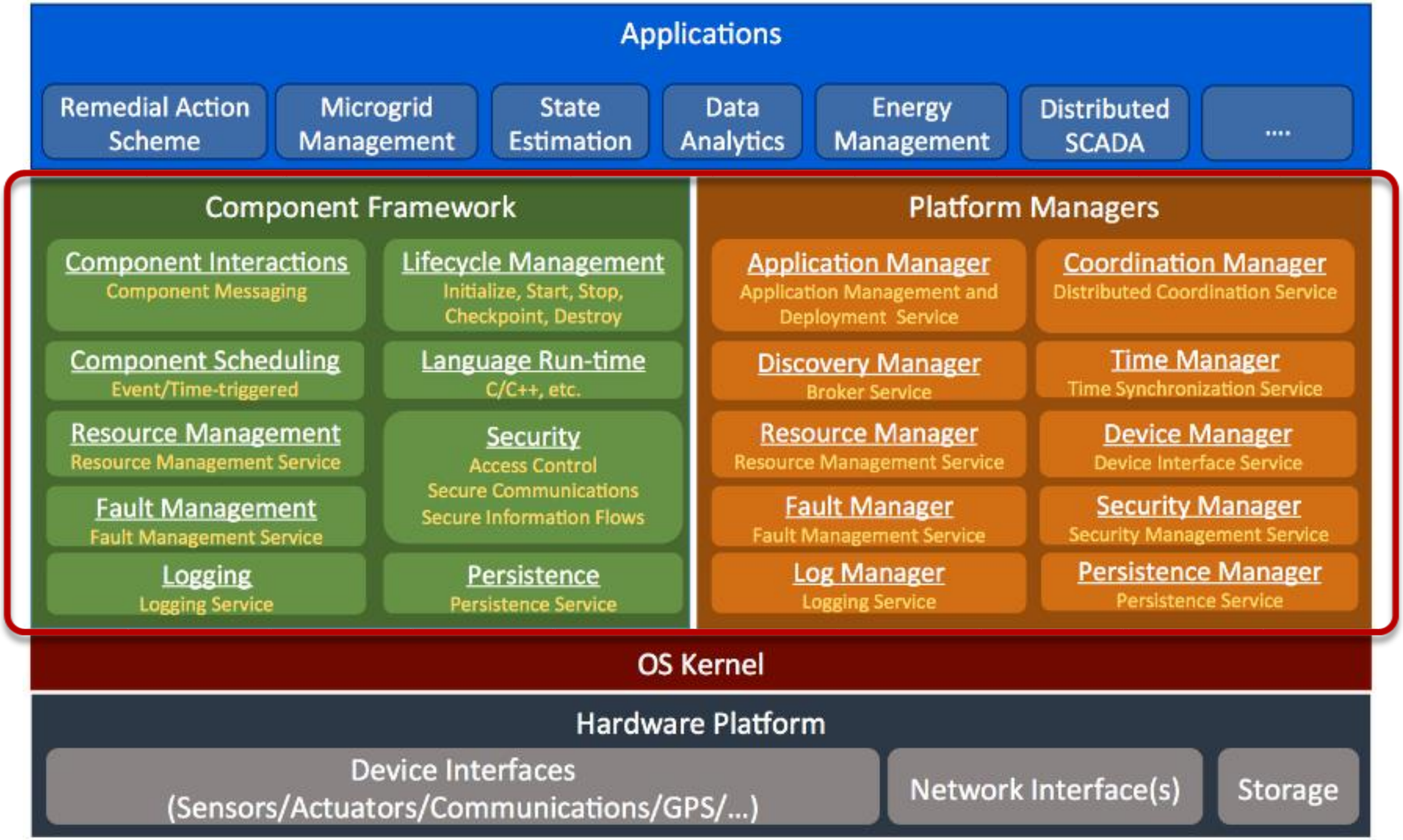
Q: IS THERE A BETTER WAY TO WRITE SOFTWARE FOR THIS?
A: YES, BUT WE NEED BETTER SOFTWARE INFRASTRUCTURE AND TOOLS.

RIAPS Vision

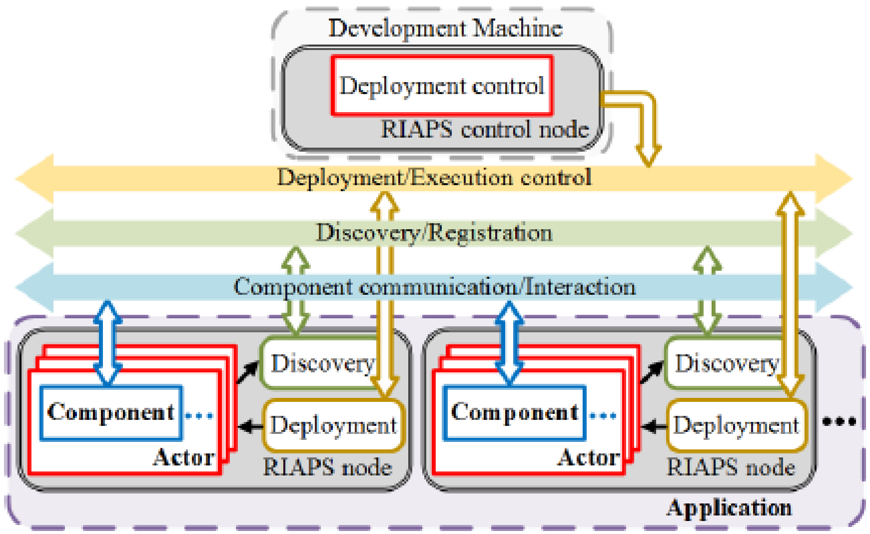


- *Showing a transmission system, but it applies to distribution systems, microgrids, etc.*

RIAPS Details
The Software Platform

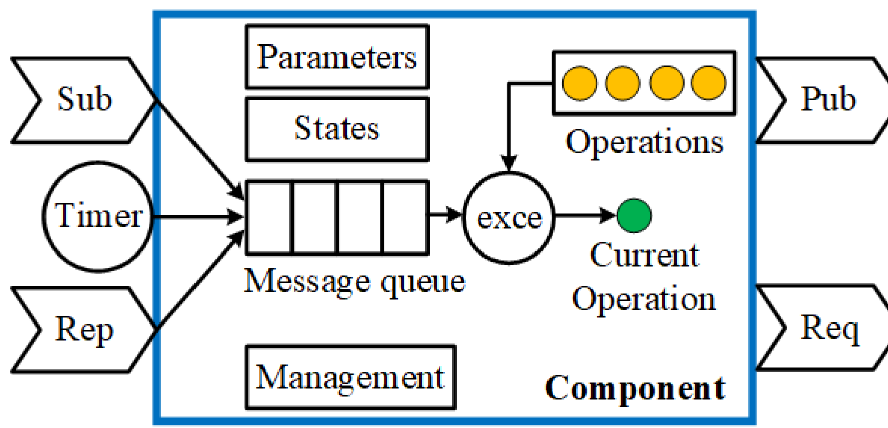


RIAPS Applications
Actors and Components



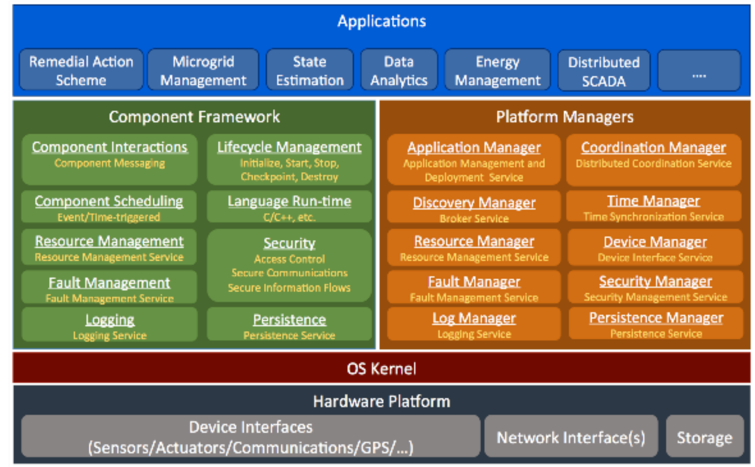
Applications consist of 'actors': distributed processes deployed on a network, serve as containers for 'components'. Actors are managed by 'deployment managers' and supported by a distributed service discovery system.

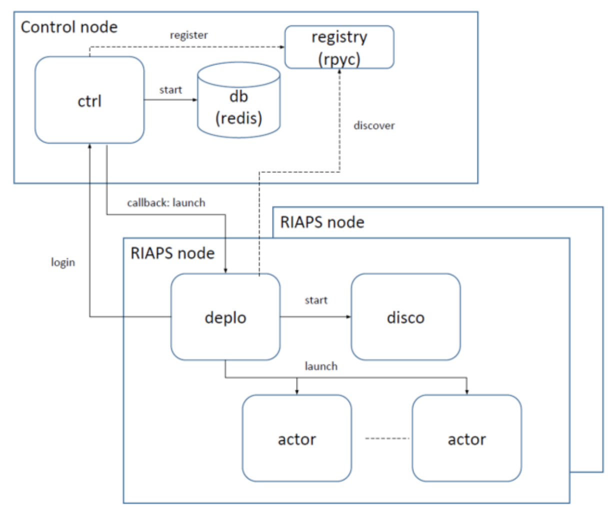
Components are (mostly) single-threaded event/time-triggered objects that interact with other components via messages. Several interaction patterns are supported.



RIAPS Platform services

- Deployment: installs and manages the execution of application actors
- Discovery: service registry, distributed on all nodes, uses a distributed hash-table in a peer-to-peer fashion
- Time synchronization: maintains a synchronized time-base across the nodes of the network, uses GPS (or NTP) as time base and IEEE-1588 for clock distribution
- Device interfaces: special components that manages specific I/O devices, isolating device protocol details from the application components (e.g. Modbus on a serial port)
- Control node: special node for managing all RIAPS nodes





RIAPS Resilience

Definition of 'Resilience' from Webster:

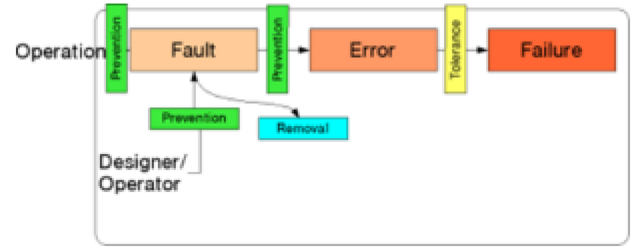
- *Capable of withstanding shock without permanent deformation or rupture*
- *Tending to recover from or adjust easily to misfortune or change*

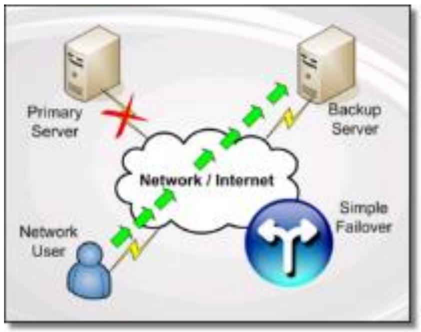
Sources of 'misfortune':

- Hardware: computing node, communication network,...
- Kernel: internal fault or system call failure,...
- Actor: framework code (including messaging layer)...
- Platform service: service crash, invalid behavior,...
- Application component faults: implementation flaw, resource exhaustion, security violation...

RIAPS
Fault management

- **Assumption**
  - Faults can happen anywhere: application, software framework, hardware, network
- **Goal**
  - RIAPS developers shall be able to develop apps that can recover from faults anywhere in the system.
- **Use case**
  - An application component hosted on a remote host stops permanently, the rest of the application detects this and 'fails over' to another, healthy component instead.
- **Principle**
  - The platform provides the *mechanics*, but app-specific behavior must be supplied by the app developer

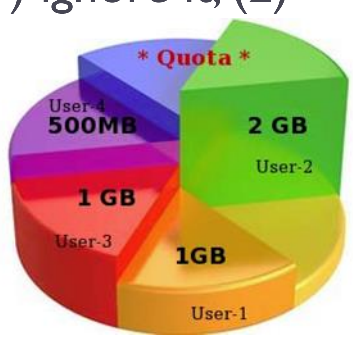




Benefit: Complex mechanisms that allow the implementation of resilient apps.

RIAPS
Resource management approach

- Resource: memory, CPU cycles, file space, network bandwidth, (access to) I/O devices
- Goal: to protect the 'system' from the over-utilization of resources by faulty (or malevolent) applications
- Use case:
  - Runaway, less important application monopolizes the CPU and prevents critical applications from doing their work
- Solution: model-based quota system, enforced by framework
  - Quota for application file space, CPU, network, and memory + response to quota violation – captured in the application model.
  - Run-time framework sets and enforces the quotas (relying on Linux capabilities)
  - When quota violation is detected, application actor can (1) ignore it, (2) restart, (3) shutdown.
    - Detection happens on the level of actors
    - App developer can provide a 'quota violation handler'
    - If actor ignores violation, it will be eventually terminated



RIAPS
Resource Models

- Resource requirements fall into 4 categories:
  - CPU requirements: a percentage of CPU time (utilization) over a given interval. If interval is missing, it defaults to 1 sec

    ```
    cpu 25% over 10 s;
    ```
  - Memory requirement: maximum total memory the actor is expected to use

    ```
    mem 512 KB;
    ```
  - Storage requirement: maximum file space the actor is expected to allocate on the file storage medium

    ```
    space 1024 KB;
    ```
  - Network requirements: amount of data expected from and to the component through the network:

    ```
    net rate 10 kbps ceil 12 kbps burst 1.2k;
    ```

RIAPS
Resource management implementation

- Architecture model specifies resource quotas
- Run-time system enforces quotas
  - Uses Linux mechanisms
- Application component is notified
  - Component can take remedial action
- Deployment manager is notified
  - Manager can terminate application actor

```
actor LimitActor {
 uses {
  cpu max 10 % over 1;//Hard limit, no 'max' is softlimit
  mem 200 mb;          // Mem limit
  space 10 mb;         // File space limit
  net rate 10 kbps ceil 12 kbps burst 1.2 k; // Netlimits
  }
  ...
}
```

RIAPS
Fault management model

Dimension	Description			Detected by
What domain does the fault occur in?	Physical	Permanent	Fail-stop (malfunction of node or cluster network)	Hardware watchdog and Platform services
			Fail-stop other (malfunction of attached device, sensor, etc.)	Device component
			Network link failure	Fault manager module in deployment manager
			Blabbering idiot (runaway publisher)	Actor/Kernel via network resource limits
		Transient	Temporary network disconnection	Fault manager module in deployment manager (network connection monitor)
			Resource exhaustion	Kernel (using resource limits)
	Cyber	Permanent	Fail-stop (actor stop due to e.g. segmentation faults)	Fault manager, using process connector from kernel)
			Resource violation	Kernel, using resource limits
			Deadline or response-time violation	Component framework logic
			Component-level anomaly	Component framework logic
		Run-time	Byzantine failures	Not handled
When does the fault occur?	Design-time		Happens while designing the system	Design tools: model-based tools, debuggers
	Deployment-time		Happens while deploying the system	Deployment manager and RIAPS Controller
	Human-caused		Unintentional mistake	Design tools and fault management architecture

Summary of results from analysis

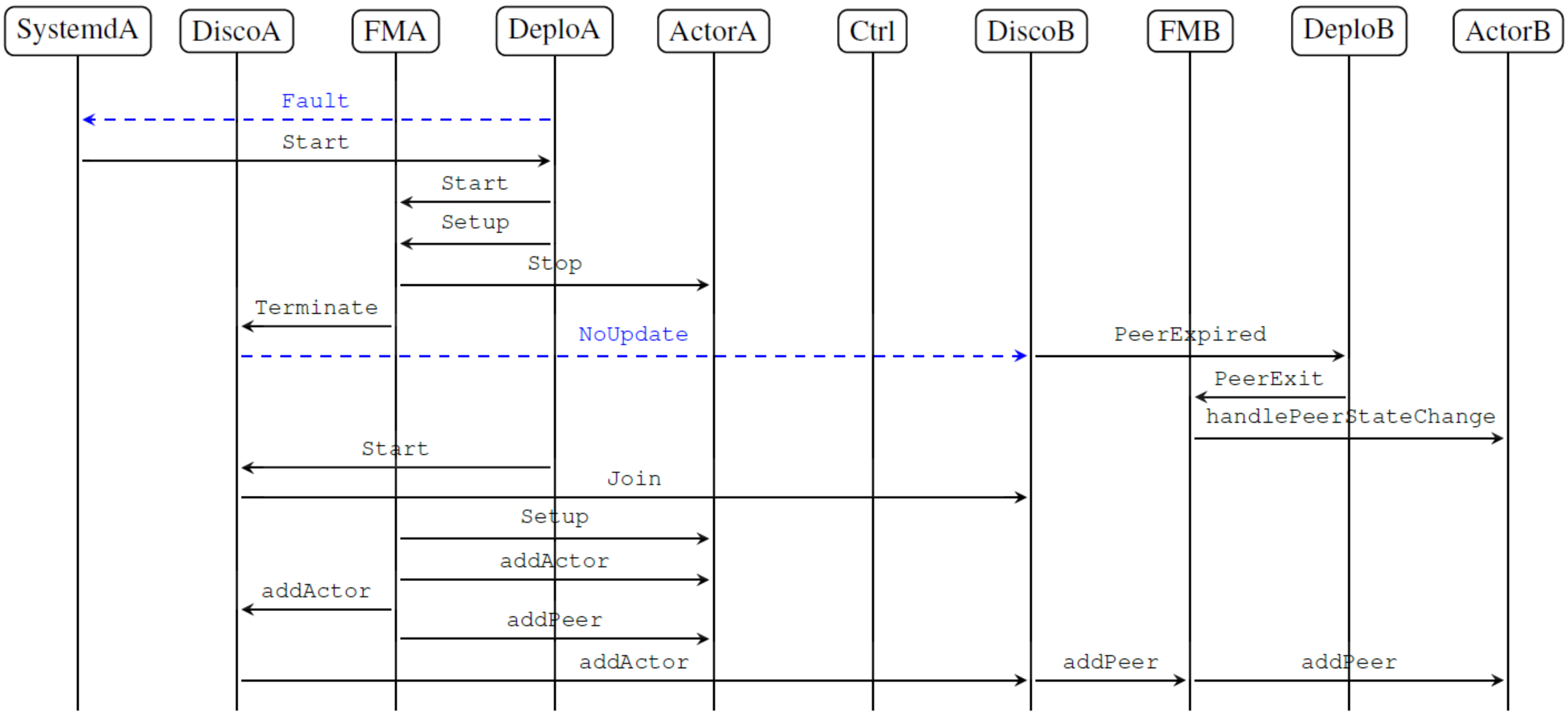
RIAPS
Fault management – Implementation (1)

Fault location	Error	Detection	Recovery	Mitigation	Tools
App flaw	actor termination	deplo detects via netlink socket	(warm) restart actor	call term handler; notify peers	libnl - lmdb as program database
	unhandled exception	framework catches all exceptions	if repeated, (warm) restart	call component fault handler; notify peers about restart	exceptions
	resource violation	framework detects	if restarted →	call app resource handler notify peers	
	- CPU utilization	soft: cgroups cpu		tune scheduler	cgroups
		hard: process monitor	if repeated, restart	notify actor/ call handler	psutil mon + SIGXCPU
	- Memory utilization	soft: cgroups memory (low)		notify actor/ call handler	cgroups + SIGUSR1
		hard: cgroups memory (critical)	terminate, restart	call termination handler	cgroups + SIGKILL
	- Space utilization	soft: notification via netlink		notify actor/ call handler	pyroute2 + quota
		hard: notification via netlink	terminate, restart	call termination handler	pyroute2 + quota
	- Network utilization	via packet stats	if repeated, (warm) restart	notify actor/ call handler notify peers about restart	nethogs
	- Deadline violation	time method calls	if repeated, restart	notify component / call handler	timer on method calls
	app freeze	check for thread stopped	terminate, restart actor	notify component; call cleanup handler; notify peers restart	threads
	app runaway	check for method non-terminating	terminate, restart actor	notify component; call cleanup handler; notify peers about restart	watchdog on method calls

RIAPS
Fault management – Implementation (2)

Fault location	Error	Detection	Recovery	Mitigation	Tools
RIAPS flaw	internal actor exception	framework catches all exception	terminate with error; warm restart	call term handler;	exceptions
	disco stop / exception	deplo detects	deplo (warm) restarts disco	if services OK, upon restart restore local service registrations	libnl + netlink
	deplo stop	systemd detects	restart deplo	(cold) restart disco ; restart local apps	Linux
	deplo loses ctrl contact	deplo detects	NIC down -> wait for NIC up; keep trying		Linux
System (OS)	service stop	systemd detects	systemd restarts	clean (cold) state	Linux
	kernel panic	kernel watchdog	reboot/restart	deplo restarts last active actors	
External I/O	I/O freeze	device actor detects	reset/start HW; device - specific	inform client component	watchdog on method calls
	I/O fault	device actor detects	reset/start HW; device - specific	log, inform client component	custom check
HW	CPU HW fault	OS crash	reset/reboot	systemd → deplo	Linux
	Mem fault	OS crash	reboot	systemd → deplo	Linux
	SSD fault	filesystem error	reboot/fsck	systemd → deplo	Linux
Network	NIC disconnect	NIC down		notify actors/call handler	pyroute2 + libnl
	RIAPS disconnect	framework detects RIAPS p2p loss	keep trying to reconnect	notify actors/call handler ; recv ops should err with timeout, to be handled by app	
	DDoS	deplo monitors p2p network performance		notify actors/call handler	netfilter + iptables

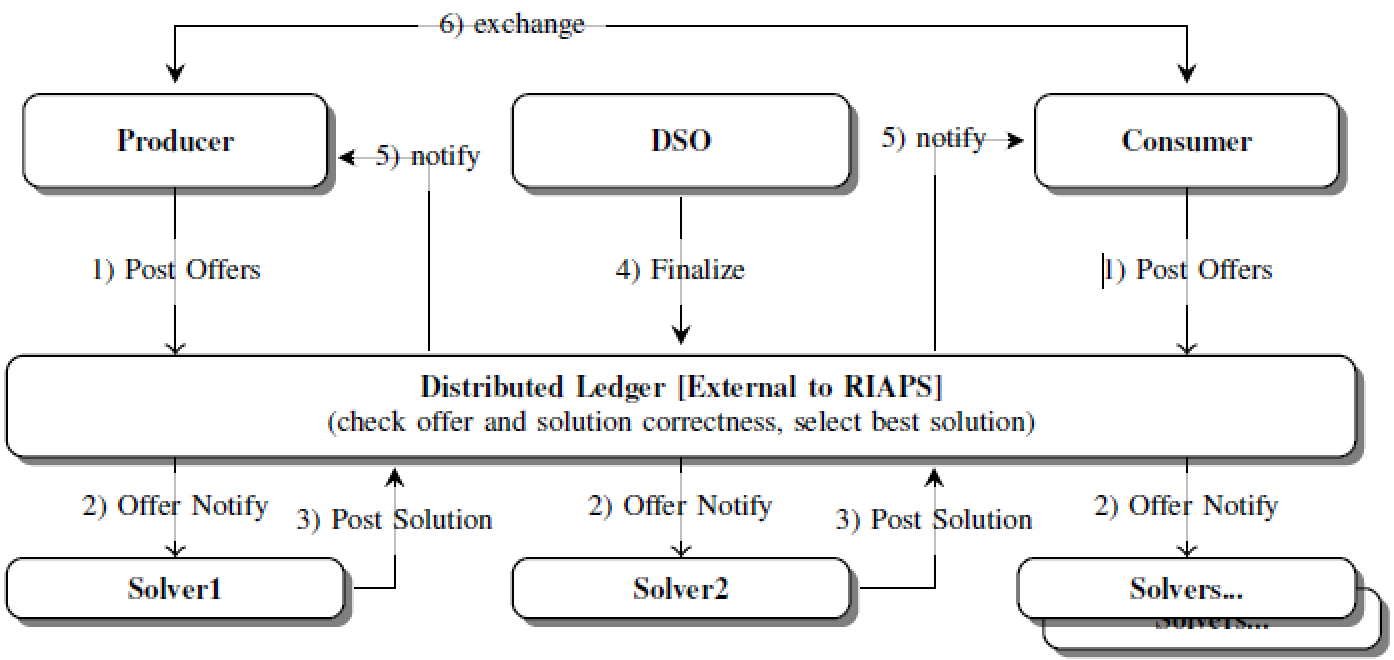
RIAPS Fault Management
Interaction patterns – notional example



Two nodes: A and B, with their own Deployment Managers (DeploA, DeploB), discovery service instances (DiscoA, DiscoB), and application actors (ActorA, ActorB). Actors of the same app form a peer-to-peer network, whose members are notified upon membership changes.

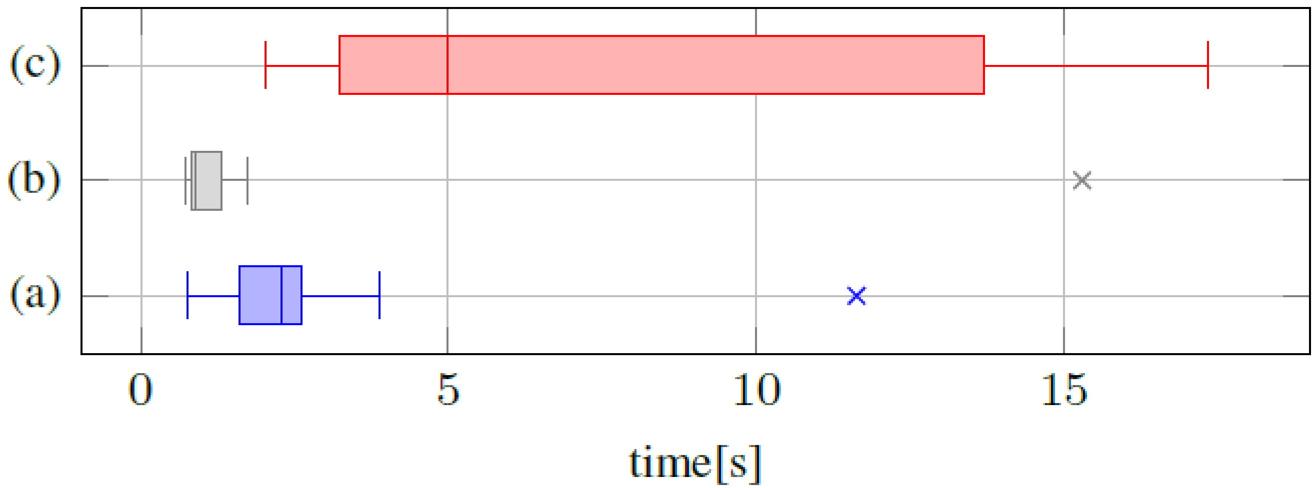
Experimental evaluation

- Transactive Energy application:
  - Prosumers 'trade energy' with the help of a 'market'. Results (trades) are recorded in a distributed ledger



Experimental evaluation

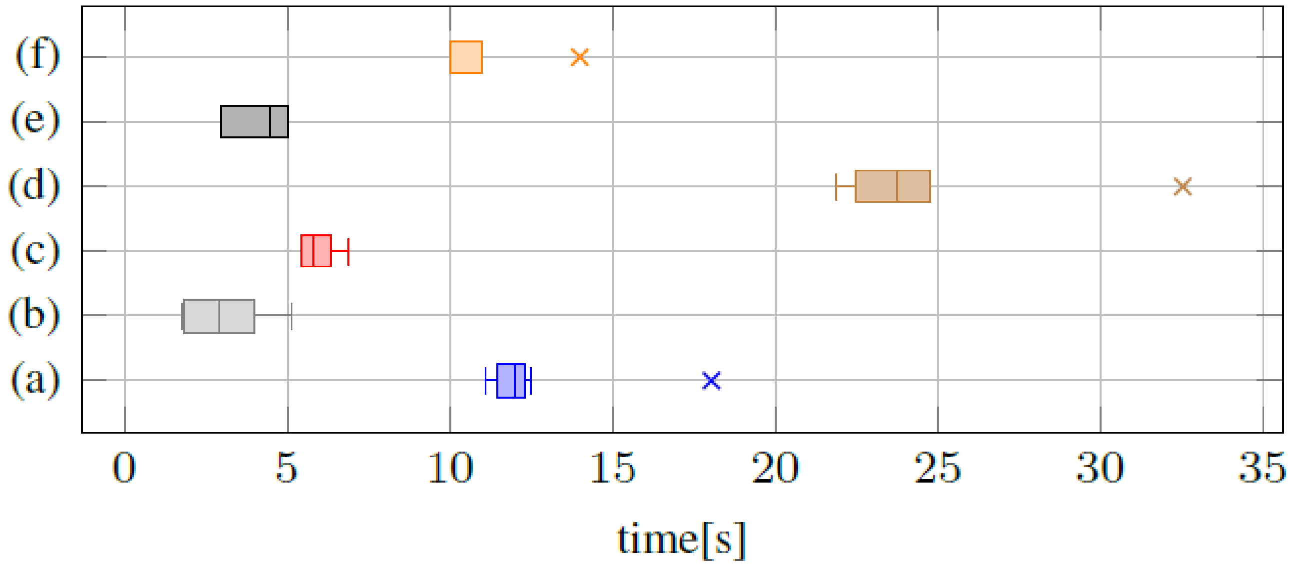
- Experiment 1: Network failure
  - Node gets disconnected, peers get notified – when node is reconnected peers are notified again



The horizontal axis is the elapsed time in seconds. The values shown from bottom to top are: (a) time to notify peer of disconnect, (b) time to notify peer of reconnect, (c) time for actors on a disconnected node to be notified.

Experimental evaluation

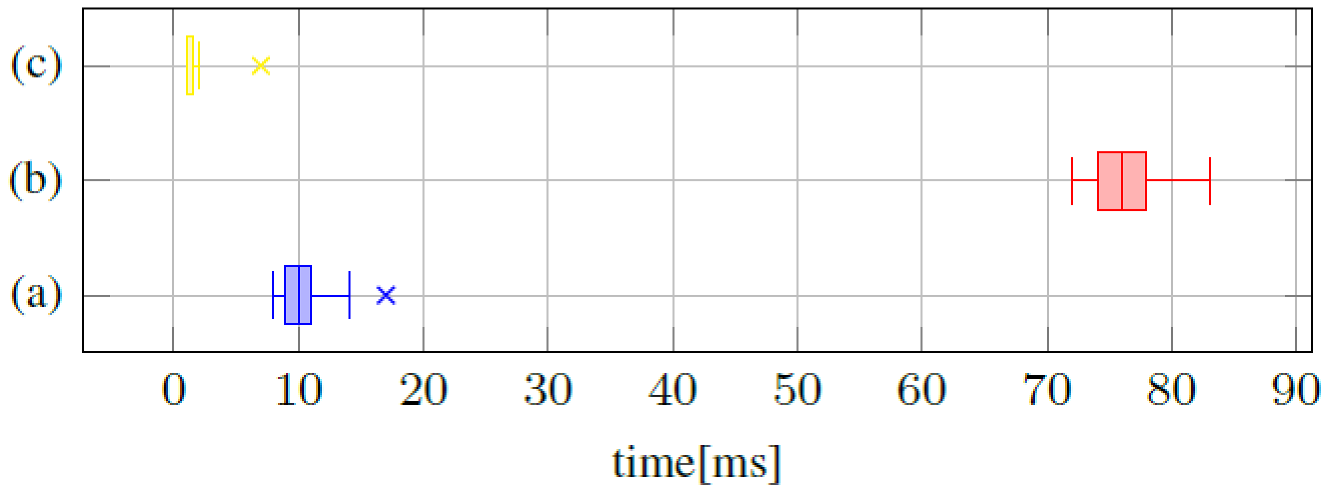
- Experiment 2: Platform failure
  - Deployment service fails



The horizontal axis is the elapsed time in seconds. From bottom to top the values are: (a) time to notify local actors, (b) time to terminate actors owned by previous deplo, (c) time to clean up other actors owned by previous deplo, (d) time until actors are fully recovered. (e) is the time until the peers know the node has left, and (f) is when the nodes know the peer has rejoined.

Experimental evaluation

- Experiment 3: Resource violations
  - CPU utilization limit, memory limit, disk space limits are exceeded



The horizontal axis is the elapsed time in milliseconds. The values indicate the time interval between detection and invocation of the associated handler for (a) Disk Usage limit, (b) Memory Usage limit and (c) CPU limit

Summary and conclusions

- RIAPS: A software platform for building distributed real-time embedded apps for the Smart Grid
- Provides a number of services to build resilient apps, including resource and fault management
- Design principles:
  - Resource management: enforce the quotas, but notify app
  - Fault management: detect and, to the extend possible, mitigate faults, but inform the app
- Websites:
  - https://riaps.isis.vanderbilt.edu/ - Project
  - https://github.com/RIAPS - Code base
  - https://riaps.github.io/ - Documents
  - https://www.youtube.com/channel/UCwfT8KeF-8M7GKhHS0muawg - Youtube channel