Designing self-manageable protocols for dependable distributed systems: an experience report

Raimundo J de A Macêdo
Distributed Systems Laboratory (LaSiD)
DCC/UFBA
www.lasid.ufba.br

IFIP 10.4 Working Group Meeting
Visegrád, Hangary, June 29, 2013

Presentation outline

✓Why self-management is needed?

✓A classification for self-manageable distributed systems

✓ Self-management for dependability

✓Final remarks

Why self-management is needed?

Computer systems are becoming more dynamic and complex in many ways:

- Large scale cyber-physical systems
- Data canters for cloud computing
- Large scale web based e-commerce systems, etc.

Such systems have to cope with

- Component failures
- Changes in the computing environment
- System updates (new component versions)
- Functional changes (new user/system requirements)

They must adapt to ever changing environment and user requirements. Adaptation must be on-the-fly and without complete previous knowledge or anticipation of what may occur

➔ Feedback loop is required for sensing both the environment and system requirements and the system should adapt itself to current running conditions, from previously specified objectives

Related Areas

- Dependability (standard and/or with resilience - JC Labrie DSN 2008, A. Avižienis 10.4 IFIP Meeting, Visegrád 2013)
- Autonomic computing (systems and networking)
- Self-adaptive systems (mostly soft Eng. flavour)
- Robotics and automation
- Real-time and embedded systems (systems/hardware flavours)
- AI
- others

Each one with its own set of conferences and journals, and particular Jargons

There are many issues: heterogeneity, SLA management, monitoring, system dynamics modelling, etc.

A classification of self-manageable distributed systems

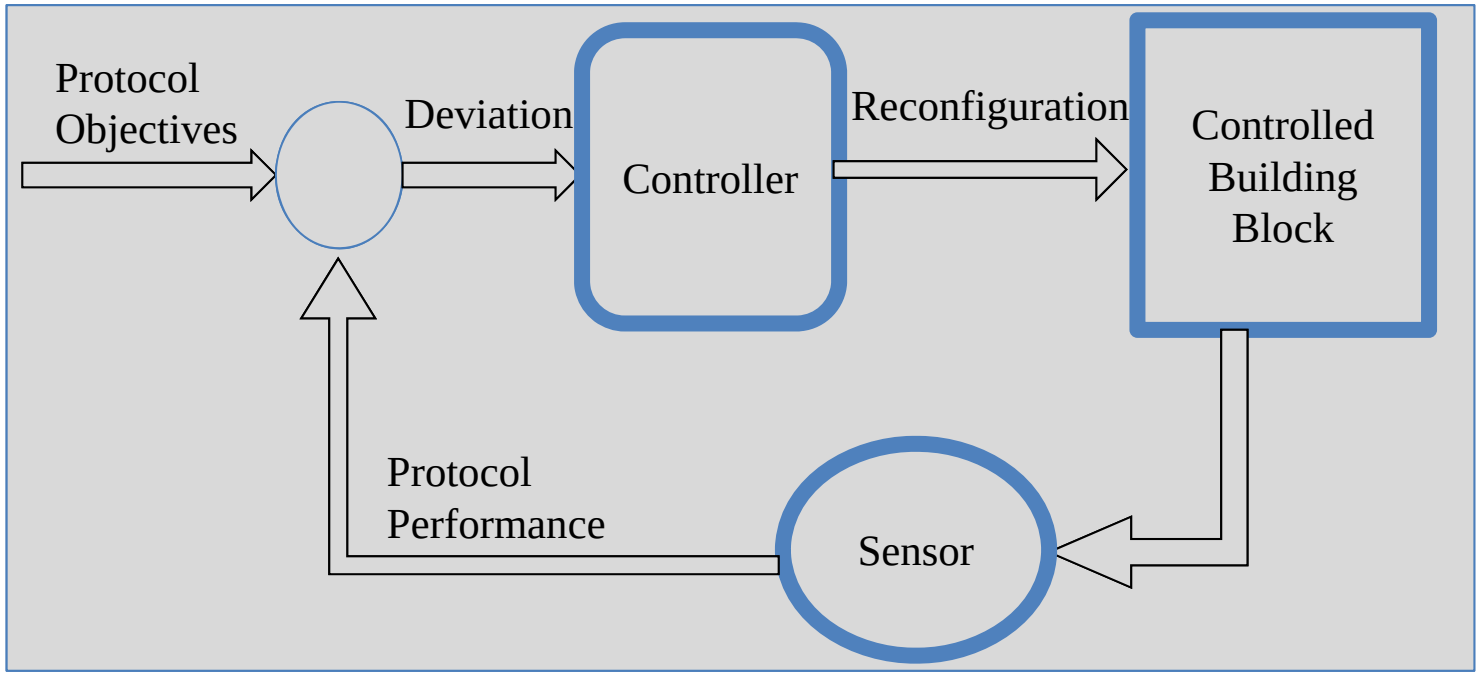
They can be classified according to the degree of control they can exercise upon the related adaptive mechanisms, if any ["***A Vision on Autonomic Distributed Systems***". Raimundo Macêdo, WoSiDA 2012, [URL:http://www.macedo.ufba.br/WoSiDA-ST3-1.pdf]

1. Non adaptive: no adaptation is provided
2. Offline adaptive: adaptation is possible but only before execution - no runtime adaptation
3. Online adaptive: adaptation at runtime from pre-defined objectives - policies cannot be controlled at runtime
4. Autonomic adaptive: adaptation policies or objectives can be modified at runtime

Self-manageable distributed systems

Self-management for dependability in distributed systems

Protocols should allow upper-layer applications or services to control their behavior from certain policies or objectives



QUESTIONS!

- ✓ What and how to specify protocols objectives ?
- ✓ How system dynamics can be modelled ?
- ✓ What components and how frequently should them be monitored?
- ✓ Where to actuate (variables) to follow protocol's objectives
- ✓ What and when to adapt the system ?

Answers depend on particular protocols and computing environments !!

A quick overview of case-studies for three commonly used building blocks for dependable distributed systems

(partially synchronous distributed system model)

- **Byzantine replication**

  Adaptive request batching for byzantine replication. Alírio Sá, Allan Freitas, and Raimundo Macêdo. Oper. Syst. Rev. 47, 1 (January 2013), pp 35-42.

- **Group Communication**

  Enhancing group communication with self-manageable behavior. Raimundo Macêdo, Allan Freitas, Alirio Sá Journal of Parallel and Distributed Computing, 73, 4 (April 2013.), pp 420-433.

- **Failure detection**

  QoS Self-configuring Failure Detectors for Distributed Systems. Alirio Sá and Raimundo Macêdo. 13th International IFIP Conference on Distributed Applications and Interoperable Systems**,** DAIS 2010, Amsterdam, The Netherlands, June 7-9, 2010. LNCC, 6115, pp 126-140.

The case for byzantine replication

Adaptive request batching for byzantine replication. Alírio Sá, Allan Freitas, and Raimundo Macêdo. Operating Systems. Rev. 47, 1 (January 2013), pp 35-42.

PBFT [Castro & Liskov] is considered by many the first successful implementation of byzantine replication. It implements a series of performance optimization mechanisms: request batching, replica rejuvenation, etc.

These mechanisms must be properly configured in order to produce the aimed performance improvements: Size and timeout for batching, checkpoint period, rejuvenation period, primary backup failure detection timeout, etc.

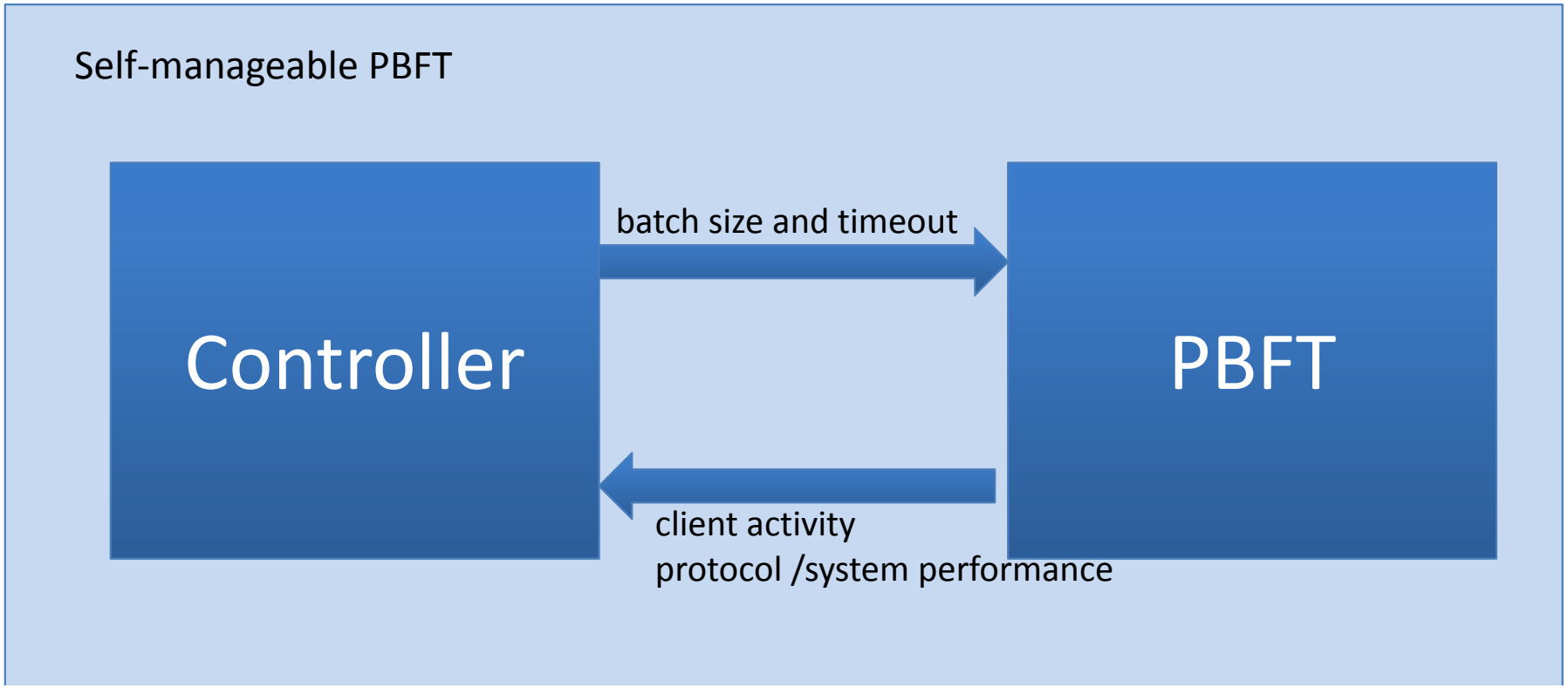
Motivation for self-manageable byzantine replication:

Automatic tuning of configuration mechanisms, based on current running conditions

We developed a self-manageable version of PBFT where the batch size and batching timeout are regulated by a controller in order to optimize message throughput and delivery time

It is online adaptive because the objective – optimizing throughput – is not modified at runtime. Online adaptive is a weaker form of a self-manageable distributed system.

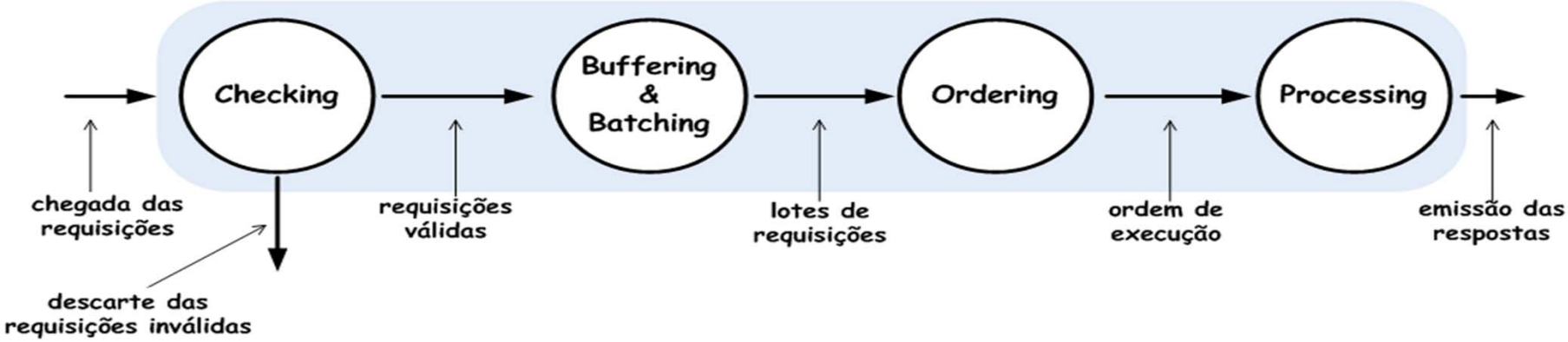
NOTE: as far as the configuration of batch size and timeout is concerned, original PBFT is offline adaptive



We abstracted PBFT as a pipeline with 4 stages:

①Message checking (to discard invalid or non-authenticated messages)
②Buffering and Batching (to handle a number of client requests)
③Ordering (to agree in a order to request processing)
④Processing (to compute and reply the requests)

Our basic idea is to keep the pipeline as busy as possible



Tradeoffs for the Batching

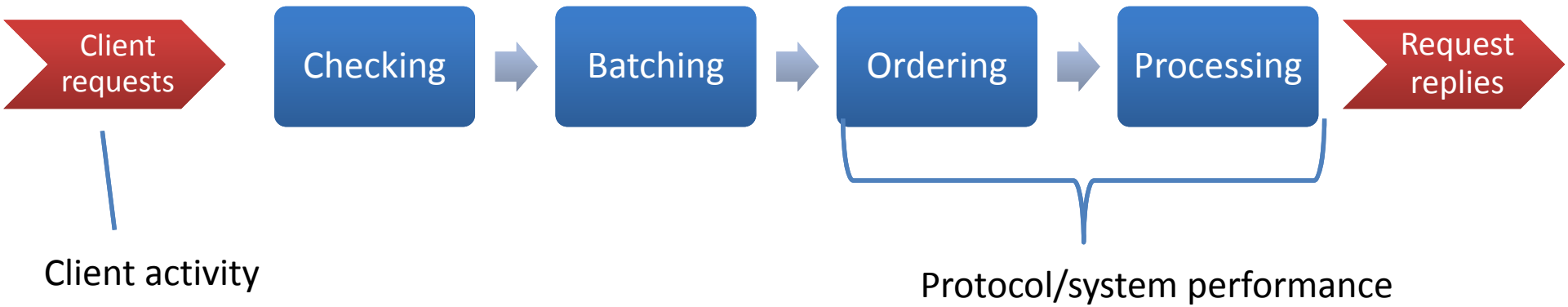
Timeout

- must be long enough to allow for the grouping of a number of requests
- must be short enough to guarantee at least one batch is in the ordering stage (too long timeouts may make the ordering stage empty)

Size

- larger sizes will decrease cryptographic and ordering costs, but, on the other hand, may delay message delivery

To optimize throughput, timeout and size are periodically adjusted to current client activity and protocol/system performance



Protocol/system performance

Mean time to order and process $MTE_k = (1 - \alpha) * MTE_{k-1} + \alpha * TE_k$

Client activity

Mean time between request arrivals $MTA = \frac{t - t_0}{n_{req}}$

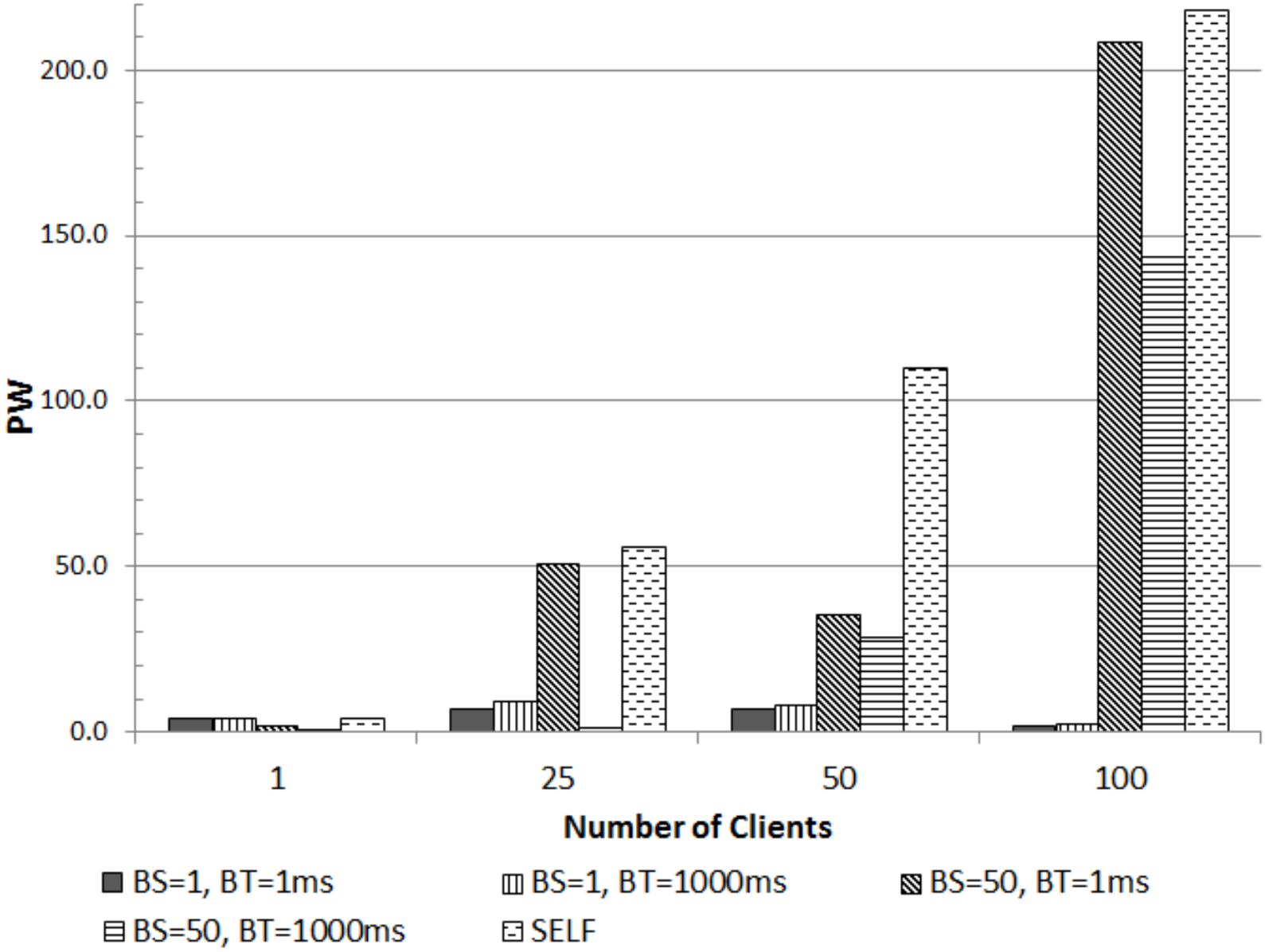
t = current time
t_0 = starting time for a primary replica r
n_{req} = number of requests

Calculating batch size (BS) and batching timeout (BT)

Periodically adjust

BT = MTE ***at least one batch is submitted after another one is processed***

$BS = \lceil \frac{MTE}{MTA} \rceil$ **If protocol is slower than request arrival, BS will increase to optimize throughput**



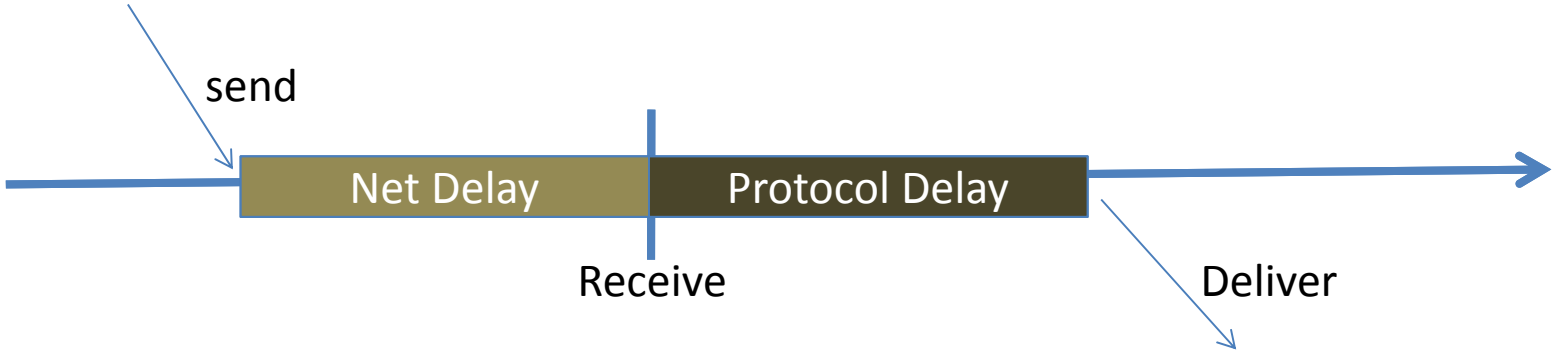
Power (PW) = throughput / delay

The case for group communication

> ***Enhancing group communication with self-manageable behavior***. Raimundo Macêdo, Allan Freitas, Alirio Sá. Journal of Parallel and Distributed Computing, 73, 4 (April 2013.), pp 420-433

Varied properties : message ordering, view synchrony etc.
Varied implementation styles : asymmetric, symmetric, logical clock based, etc.

Usual Behavior: sent messages are kept in buffers until prescribed properties can be guaranteed and/or until failures are handled



Total delivery delay depends on loads, protocol overhead, and failures

The self-manageable approach for group communication

Resource consumption abstraction as a function of message end-to-end delays

The self-manageable approach manipulates a single variable (time-silence) to control the protocol behavior (message overhead, delivery delay)

If application messages are not sent to provide block completion/stability, we have to force this block completion/stability with protocol messages:

	P_1	P_2	P_3
1	+		
2		+	
3	+		+

?

Time silence mechanism is triggered, if the process does not contribute for this completion after a period of ts

Self-Manageable Approach

General View

A. User/system defines an upper-bound for resource consumption, for the protocol

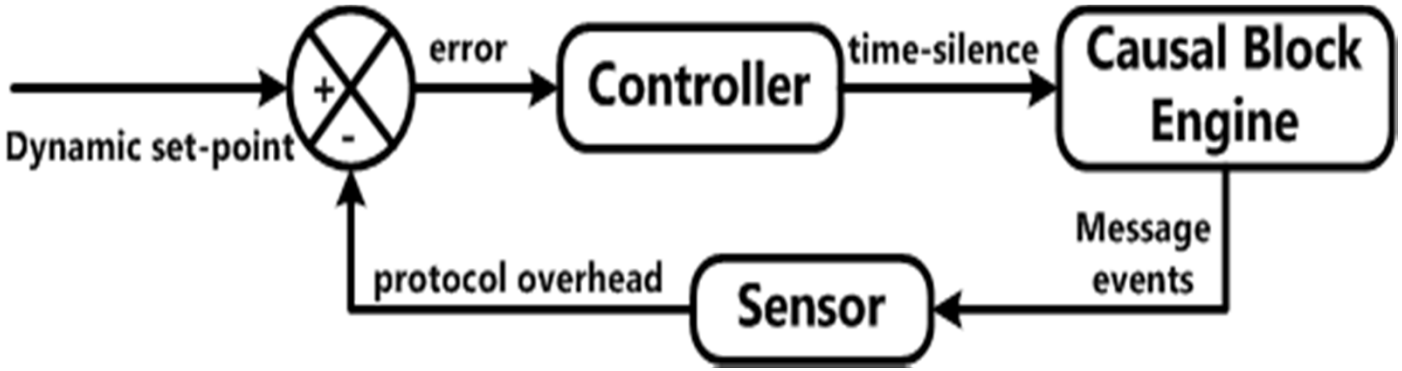
can be changed at runtime
AUTONOMIC ADAPTIVE

B. The approach computes the overhead which can be imposed by the CB protocol in such a way that it doesn't infringe the expected upper-bound (best effort)

⇒Such overhead is a dynamic set-point for the protocol and it is computed considering:

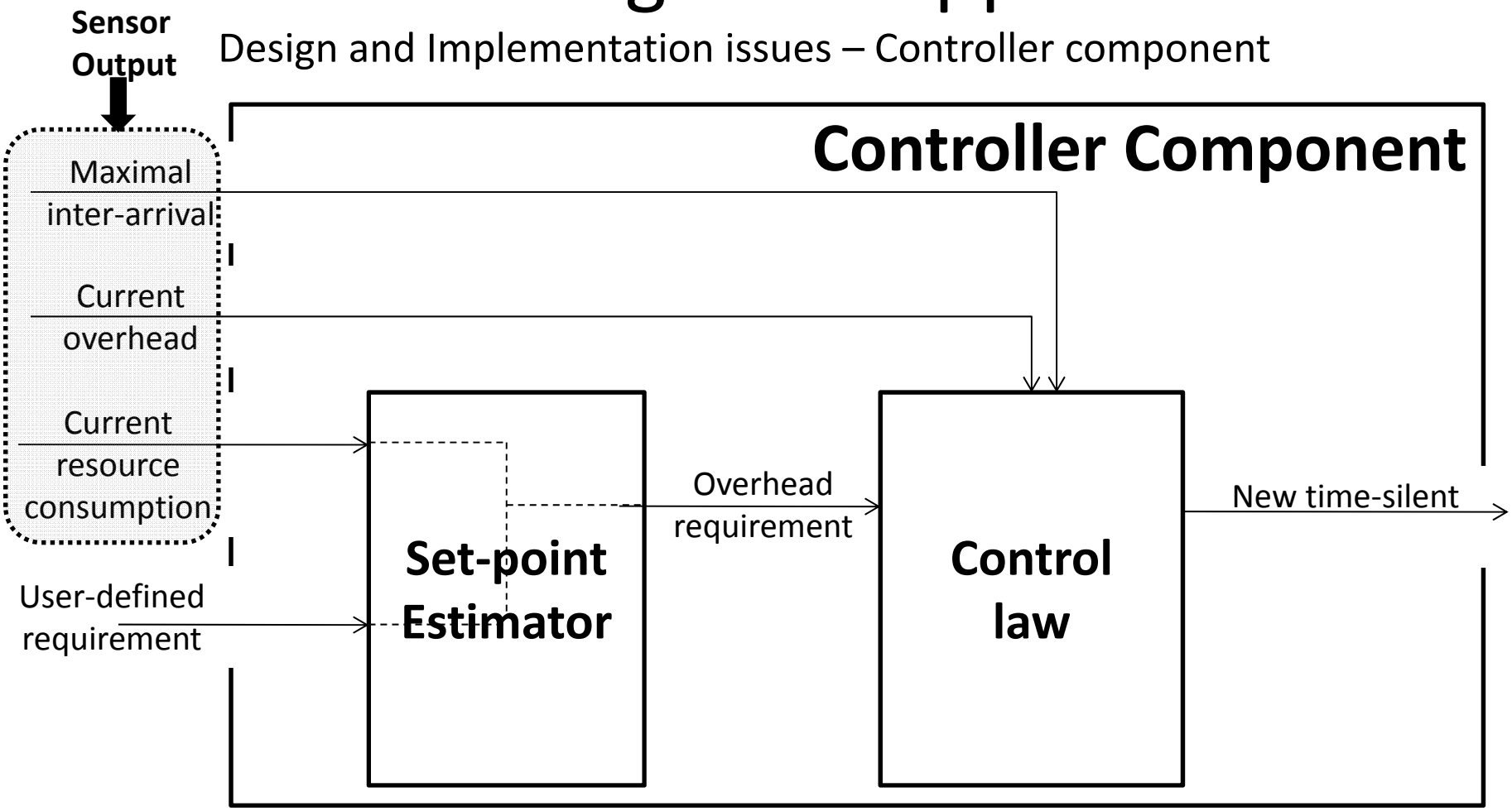
(a) User expectation;
(b) The state of the computing environment;
(c) The performance of the group communication protocol;

C. The approach defines a new operation point for the CB protocol based on the dynamic imposition for the overhead set-point.



Self-Manageable Approach

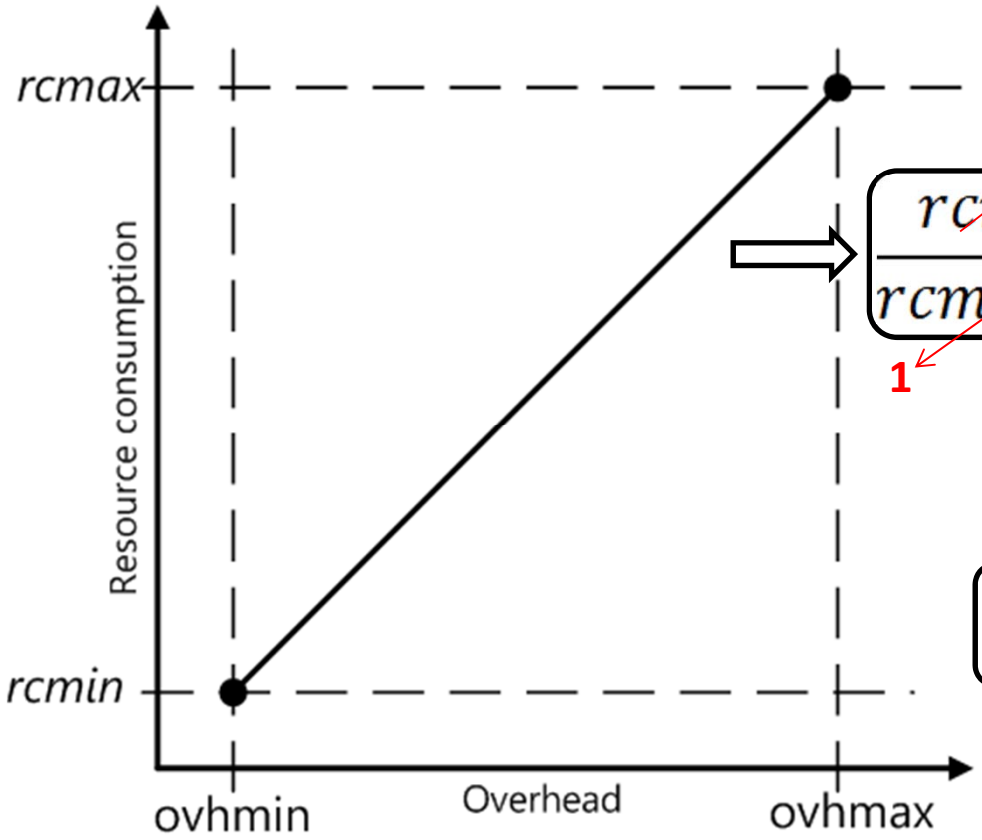
Design and Implementation issues – Controller component



⇒ The controller uses the output of the sensor component and the user-defined requirement in terms of resource consumption to estimate a new time-silent.

Self-Manageable Approach

Design and Implementation issues – Controller component



$$\frac{rcmin + \Delta rc}{rcmax - rcmin} = \frac{ovhmin + \Delta ovh}{ovhmax - ovhmin}$$

0 0 1 0 0

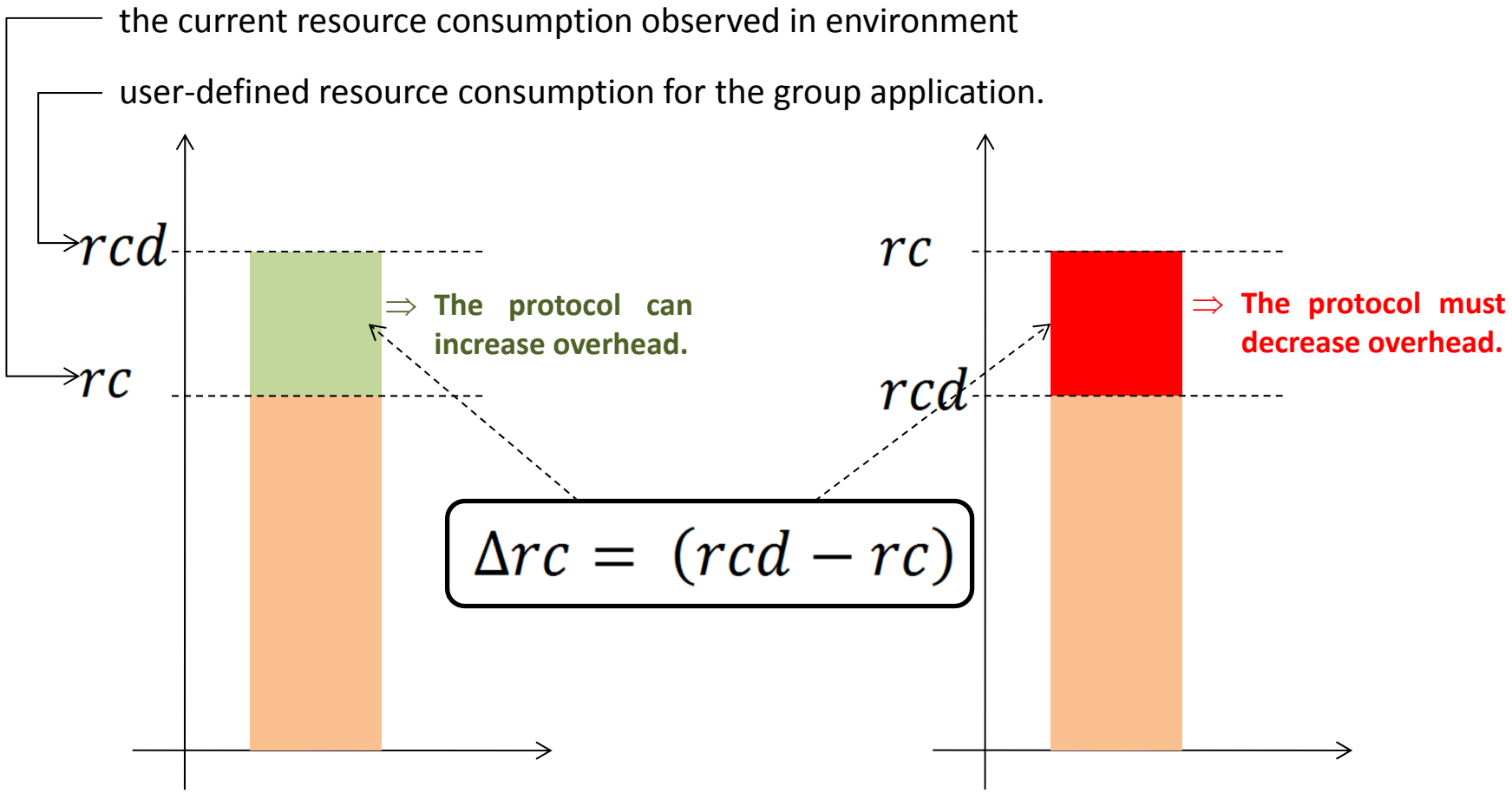
$$\Delta ovh = \Delta rc * ovhmax$$

⇒ The control design assumes a linear relationship between overhead and resource consumption.

⇒ This relationship gives to us a intuition about what a variation in resource consumption means in terms of overhead and vice-verse.

Self-Manageable Approach

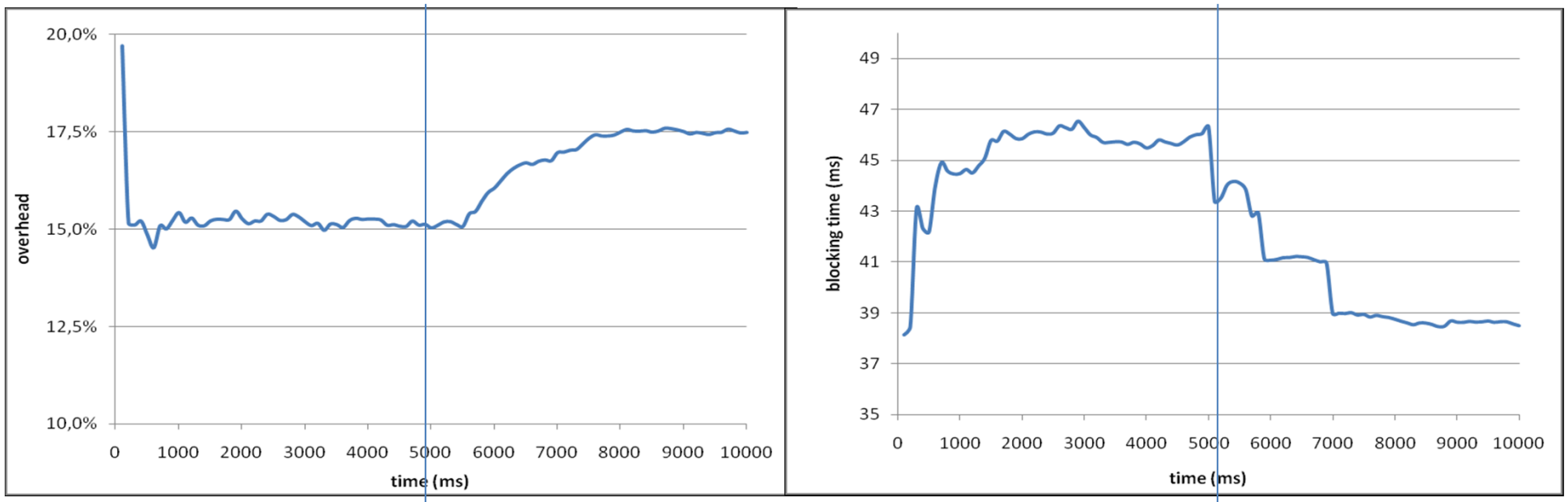
Design and Implementation issues – Controller component



Performance Evaluation

Self-Manageable adaptation For Change in Desired QoS Setup

Desired QoS (Set-point) changing at t = 5000 ms



Change on sla (rcd varies from 25% to 30%)

The case for failure detectors

QoS Self-configuring Failure Detectors for Distributed Systems. Alirio Sá and Raimundo Macêdo. 13th International IFIP Conference on Distributed Applications and Interoperable Systems**,** DAIS 2010, Amsterdam, The Netherlands, June 7-9, 2010. LNCC, 6115, pp 126-140.

Trade-offs in the configuring of a failure detector (detection timeout and monitoring periods):

- Larger detection timeout and/or larger monitoring periods imply on larger detection latency.
- Shorter detection timeout can imply on more detection mistakes.
- Shorter monitoring periods implies on higher consumption of the computing resources and unreliable detections in cases of exhaustion of the computing resources (processing and communication).

- **CHALLENGE**
  - Build a failure detector that self-adjust its operating parameters due to changes on the computing environment or application requirements - from predefined QoS parameters

- **BASIC DIFFICULTY**

  The environment characteristics are unknown and can change over time.

  - Load variations, changes in available computing resources, faults etc.

  ➔The dynamic behavior of the distributed environment is hard to be characterized by specific probability distributions.

Our self-manageable approach (autonomic adaptive)

Environment and Detection Service Sensing

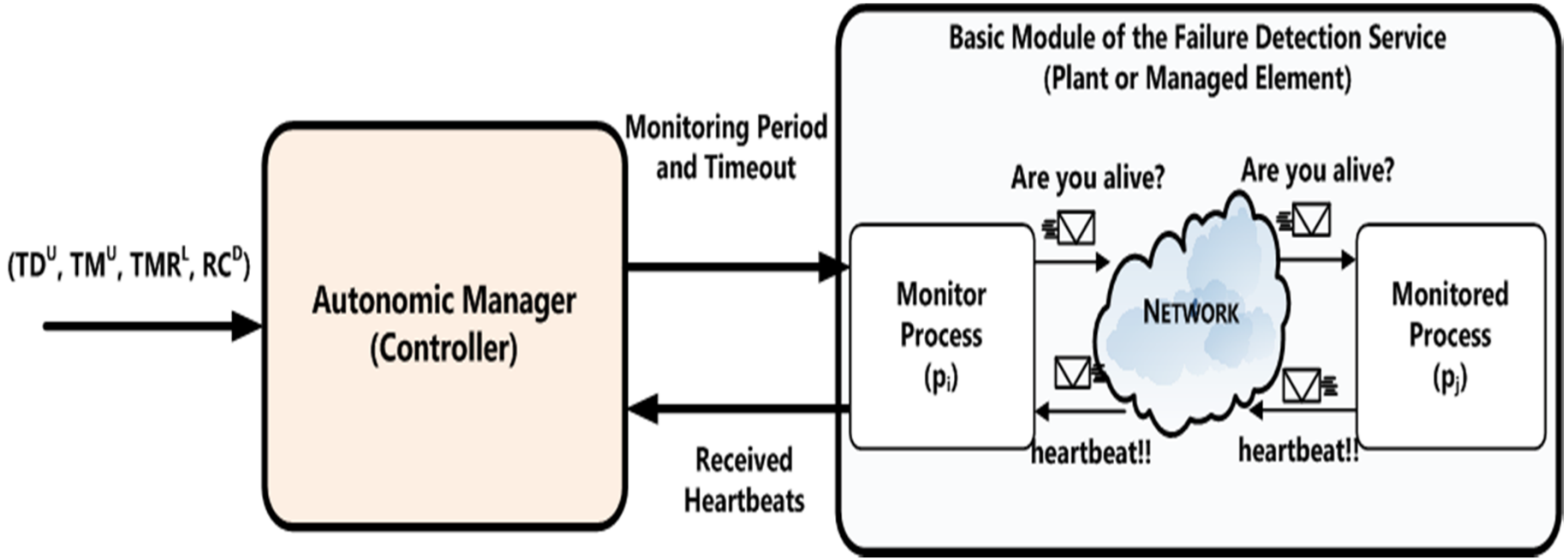
Observes variables related to environment behavior and quality of detection delivered.

Detection Timeout Regulation

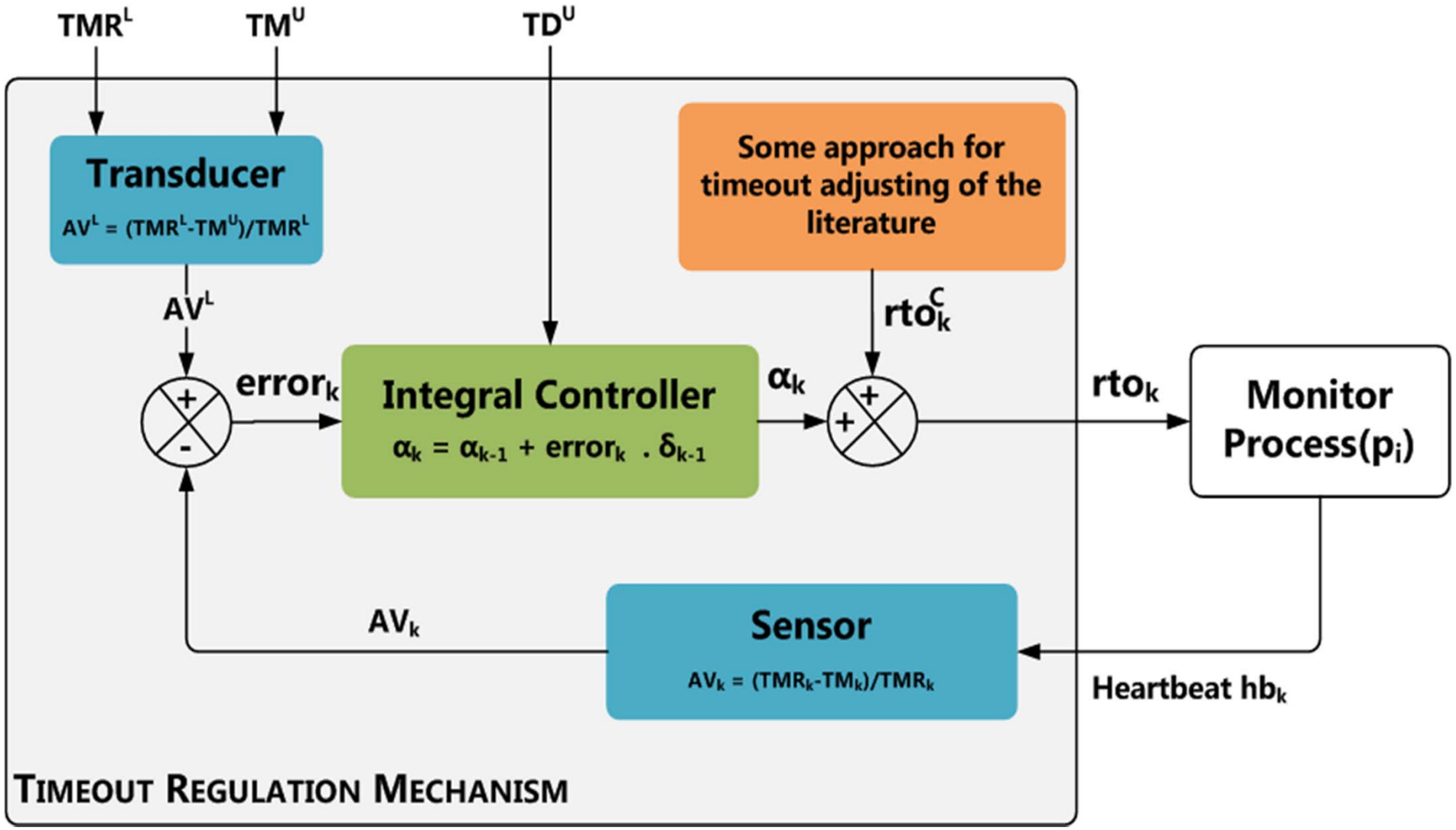
Adjust the detection timeout to allow more reliable detection with lower impacts on detection time.

Monitoring Period Regulation

Adjust in the monitoring period to allow a faster detection with lower impact for other applications which share the computing environment**.**



Timeout Regulation



Monitoring period regulation is a little bit more complicated (see the paper)

Final words

- A challenge in designing self-manageable DS is how to map business rules and policies into protocol objectives. This mapping greatly depends on mastering related DS building block dynamics and its performance tradeoffs.

- Modeling the behavior of the computing environment and distributed system protocol is another challenge, which we have addressed (though, to a limited extend) by applying feedback control loop theory.

- We are applying the concepts and protocols presented here in the construction of an autonomic dependability manager to a federated cloud system (Project JIT CLOUDS – CTIC/RNP)

Thanks !!