Feb 20

FALCON and RAPID

Vijay Chidambaram

Detecting Failures

- Detecting failures in distributed systems is important
- Often, detecting failures takes longer than the recovery
- Falcon provides:
    - sub-second detection in the common case
    - No false positives
    - Terminates minimum needed component to ensure results

Time outs in real systems

- Real time-outs:
  - Google File System: 60 seconds
  - Chubby: 12 seconds
  - Dryad: 30 seconds
  - Network File System: 60 seconds

The problem

- Picking time outs correctly is hard
- Achieving a failure detector with three properties simultaneously is hard:
    - Quick detection
    - Reliability
    - Minimal disruption

Reliability

- If the failure detector only cared about reliability, it could kill any process it suspects to be down
    - Therefore making sure in the process
- This is formally called STONITH: Shoot the Other Node in the Head

FALCON

- Fast And Lethal Component Observation Network (FALCON) :)
- Good properties for a Failure Detector:
  - If a node is up, it should be reported as up
  - When a node is down, it should be reported as down after a while
  - The gap to detect down nodes should be low
  - The failure detector should not cause disruption

Core Insight

- Failures are observed quickly if we look at the right software layer
- A process that core dumps will disappear from the process table
- after an operating system panics, it stops scheduling processes
- if a machine loses power, it stops communicating with its attached network switch.
- If the failure detector infiltrates various layers in the system, it can provide reliable failure detection using local instead of end-to-end timeouts and sometimes without using any timeouts.

FALCON

- Spy modules are inserted within various software layers
- If a spy suspects a process is down, it KILLS it
- Thus, we have quick detection and reliability
- Handles crashes of all nodes, but not handle byzantine failures

FALCON

- But spies are embedded within layers, so if a layer goes down, so does a spy
- How do we handle this?
- Answer: we make spies monitor other spies

FALCON Spy Network

- Spies are arranged in a chained network
- Spy monitors the spy in layer above
- OS spy monitors the application spy
- FALCON assumes that spies will miss detecting some failures: backup is a large time-out
- If a network partition happens, FALCON is paused (can't communicate with remote spies)

FALCON Architecture

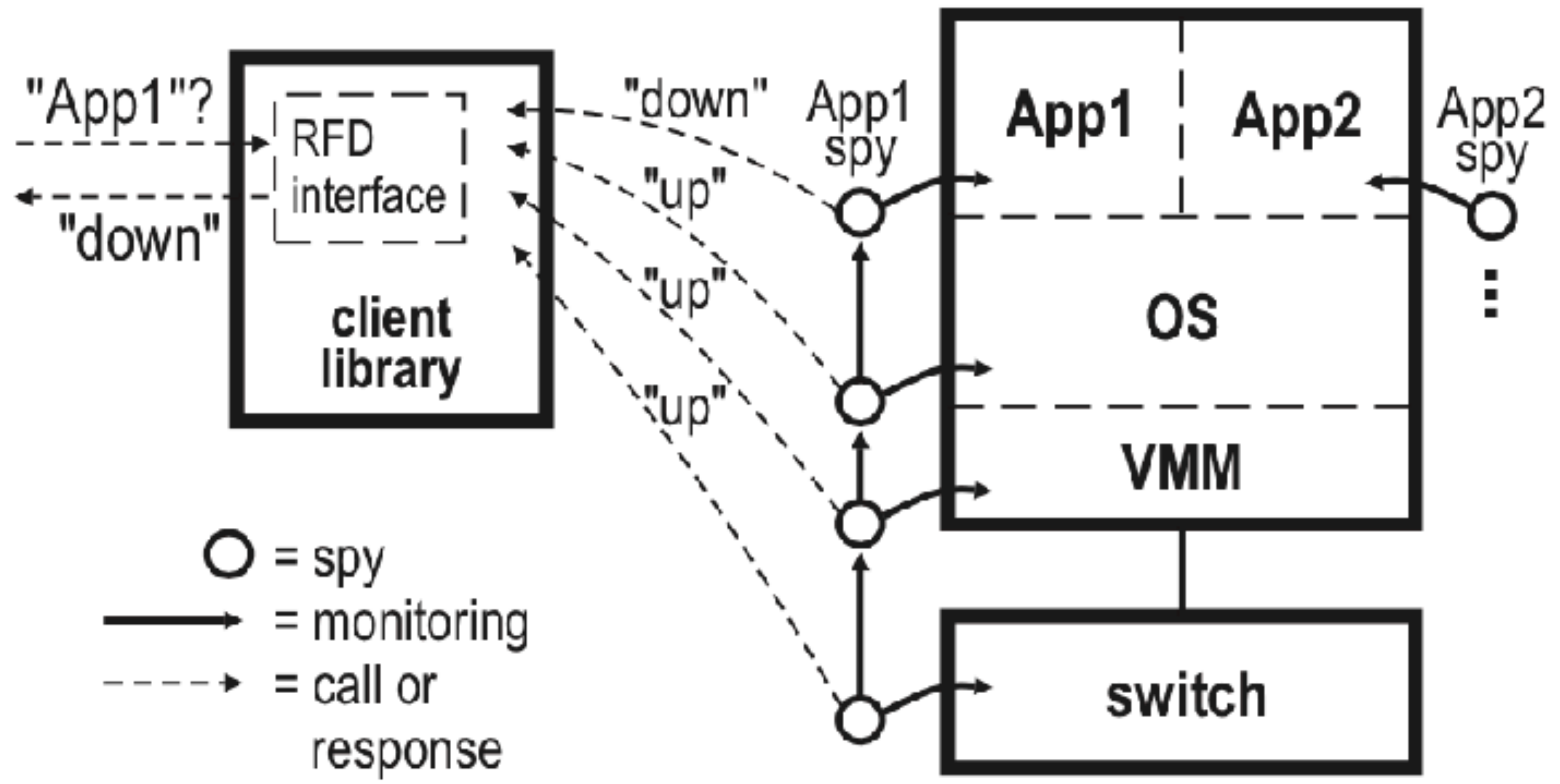
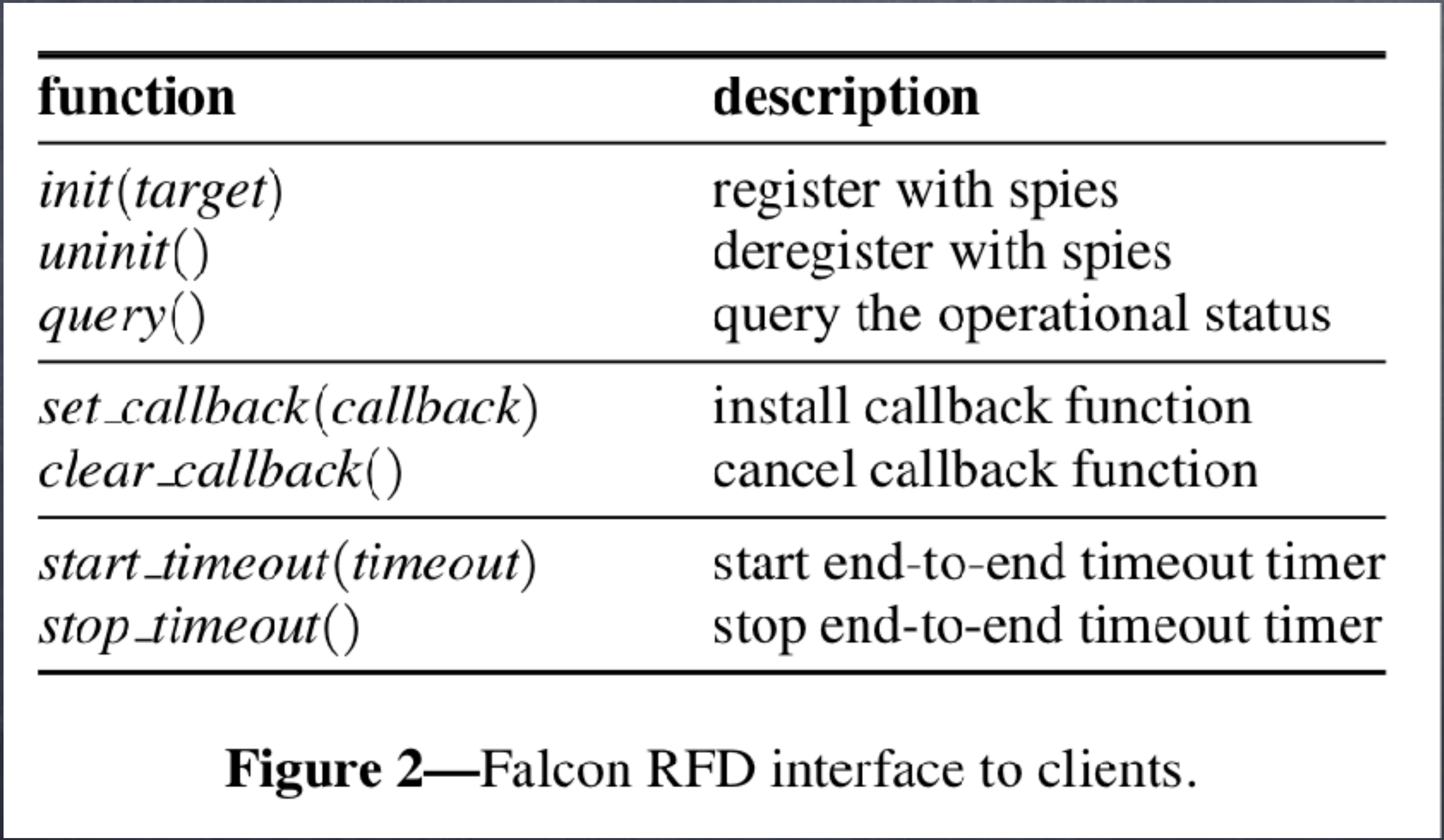


Figure 1—Architecture of Falcon. The application spy provides accurate information about whether the application is up; this spy is the only one that can observe that the application is working. The next spy down provides accurate information not only about its layer but also about whether the application spy is up; more generally, lower-level spies monitor higher-level ones.

FALCON Interface



function	**description**
init(*target*)	register with spies
uninit()	deregister with spies
query()	query the operational status
set_callback(*callback*)	install callback function
clear_callback()	cancel callback function
start_timeout(*timeout*)	start end-to-end timeout timer
stop_timeout()	stop end-to-end timeout timer

Figure 2—Falcon RFD interface to clients.

FALCON Spy Interface

- Register() - set callback to client
- Cancel() - cancel callback
- Kill() - kill monitored layer

Implementation

- Spies at four layers: OS, Application, VMM, network switch
- Integrated into an application with tens of lines of code

Results

- Added Falcon to ZooKeeper
  - Increases availability by 6x
- Added Falcon to replication library
- Falcon simplifies applications since they can build on top of reliable failure detection

RAPID

- Real-world failures are more than crashes:
  - misconfigured firewalls
  - one-way connectivity loss
  - flip-flops in reachability
  - some-but-not-all packets being dropped
- Existing tools take too long to converge to stable state

The problem

- Detecting membership in a stable manner
- When membership, cluster does heavy-duty operations such as moving data around
  - We want to keep this to a minimum
-

Existing membership schemes

- Logically centralized services
- Gossip based services
  - Can diverge!
- Census: gossip + consistency

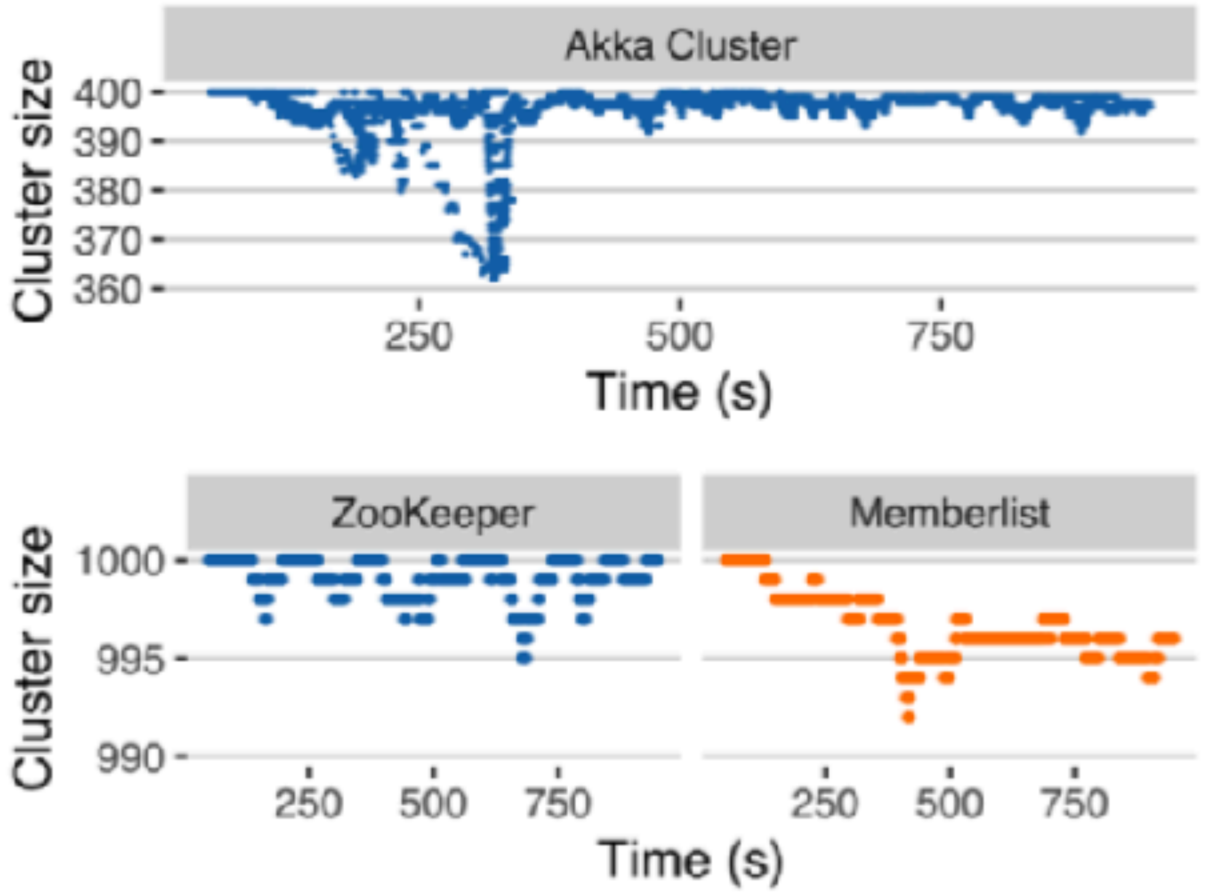


Figure 1: Akka Cluster, ZooKeeper and Memberlist exhibit instabilities and inconsistencies when 1% of processes experience 80% packet loss (similar to scenarios described in [19, 49]). Every process logs its own view of the cluster size every second, shown as one dot along the time (X) axis. Note, the y-axis range does not start at 0. X-axis points (or intervals) with different cluster size values represent inconsistent views among processes at that point (or during the interval).

RAPID Overview

- Have a set of observer processes
- Arrange them as part of an expander overlay graph
- A subject is monitored by multiple observers
- Multiple observers indicating failure is taken as high-fidelity signal
- Observers detect a cut: multiple nodes that must be dropped from membership
- RAPID waits for multiple observers to detect same cut

RAPID

- A configuration in Rapid comprises a configuration identifier and a membership-set (a list of processes).
- Every process p (a subject) is monitored by K observer processes. If L-of- K correct observers cannot communicate with a subject, then the subject is considered observably unresponsive.
-

Topology

- Rapid organizes processes into a monitoring topology that is an expander graph
- We use the fact that a random K-regular graph is very likely to be a good expander for K ≥ 3
- We construct K pseudo- randomly generated rings with each ring containing the full list of members.
- A pair of processes (o, s) form an observer/subject edge if o precedes s in a ring

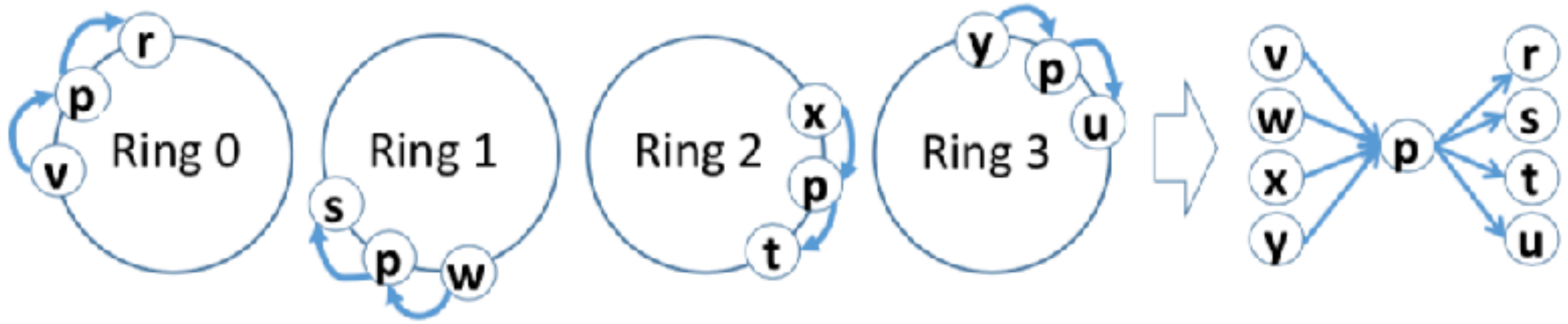


Figure 2: *p*'s neighborhood in a $K = 4$-Ring topology. *p*'s observers are $\{v, w, x, y\}$; *p*'s subjects are $\{r, s, t, u\}$.

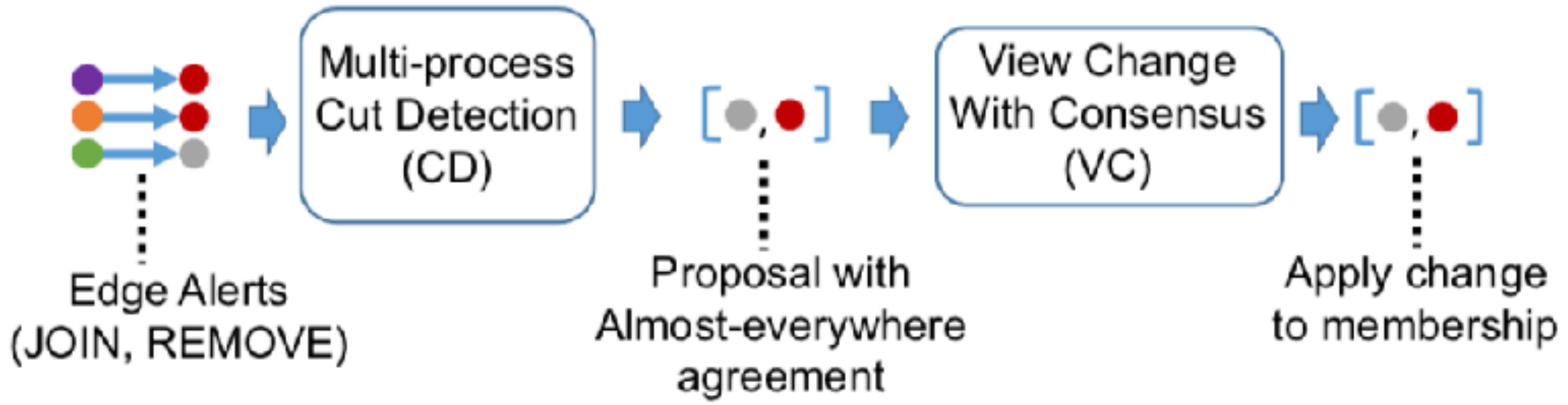


Figure 3: Solution overview, showing the sequence of steps at each process for a single configuration change.

Why this topology?

- Strong connectivity: if F out of V processes are fault, there will be (V-F/F) edges from the fault subset to the rest of the processes
- As a result, failures are detected by observers with high probability
- Every process observes K processes, and is observed by K processes. Bounds overhead to O(K)
- Every process joining or leaving results in 2*K edges being added or removed (2 edges each in K graphs)

Multi-process Cut Detection

- Each process gets many alerts
  - Alerts are aggregated until a stable multi-process cut is detected
  - A process defers a decision on a single process until the alert-count it received on all processes is considered stable.
  - In particular, it waits until there is no process with an alert-count above a low-watermark threshold L and below a high-watermark threshold H.
  - Notifications below L - noise
  - Notifications above H - stable decision about process

Proposing Config Changes

- Delay proposing a configuration change un- til there is at least one process in stable report mode and there is no process in unstable report mode

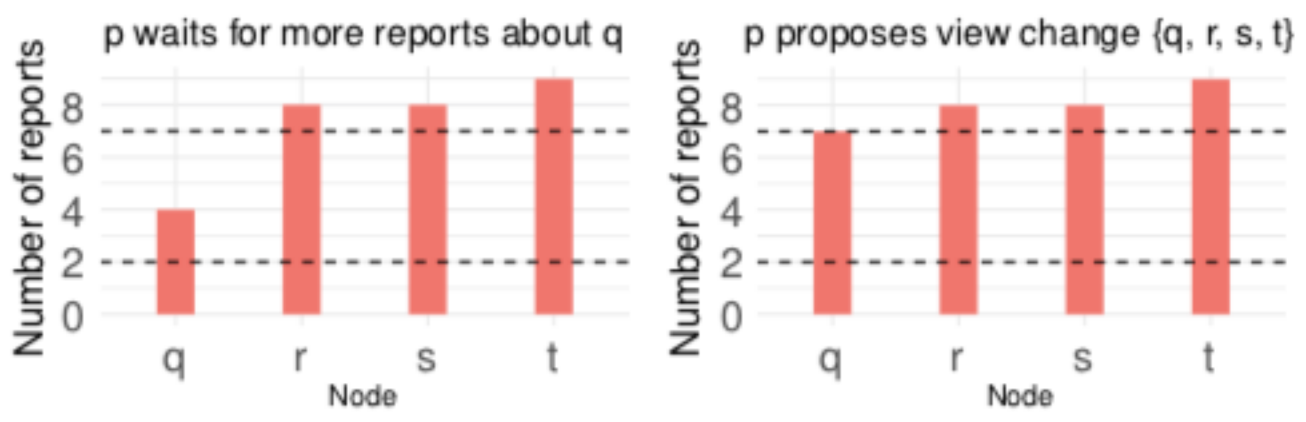
- 

Figure 4: Almost everywhere agreement protocol example at a process p, with tallies about q,r,s,t and $K = 10, H = 7, L = 2$. K is the number of observers per subject. The region between H and L is the unstable region. The region between K and H is the stable region. **Left:** $stable = \{r,s,t\}$; $unstable = \{q\}$. **Right:** q moves from *unstable* to *stable*; p proposes a view change $\{q,r,s,t\}$.

Config Changes

- Once nodes propose a config change, it is applied using a consensus algorithm
- RAPID uses Fast Paxos
- The proposed config change is used as input
- If two thirds of the processes have identical input, Fast Paxos reaches a decision
- Due to the almost-everywhere agreement in RAPID, Fast Paxos reaches decisions quickly

Results

- RAPID brings up a 2000 node cluster 2-5.8x faster than ZooKeeper or Memberlist
- Robust to node crashes, network partitions, etc

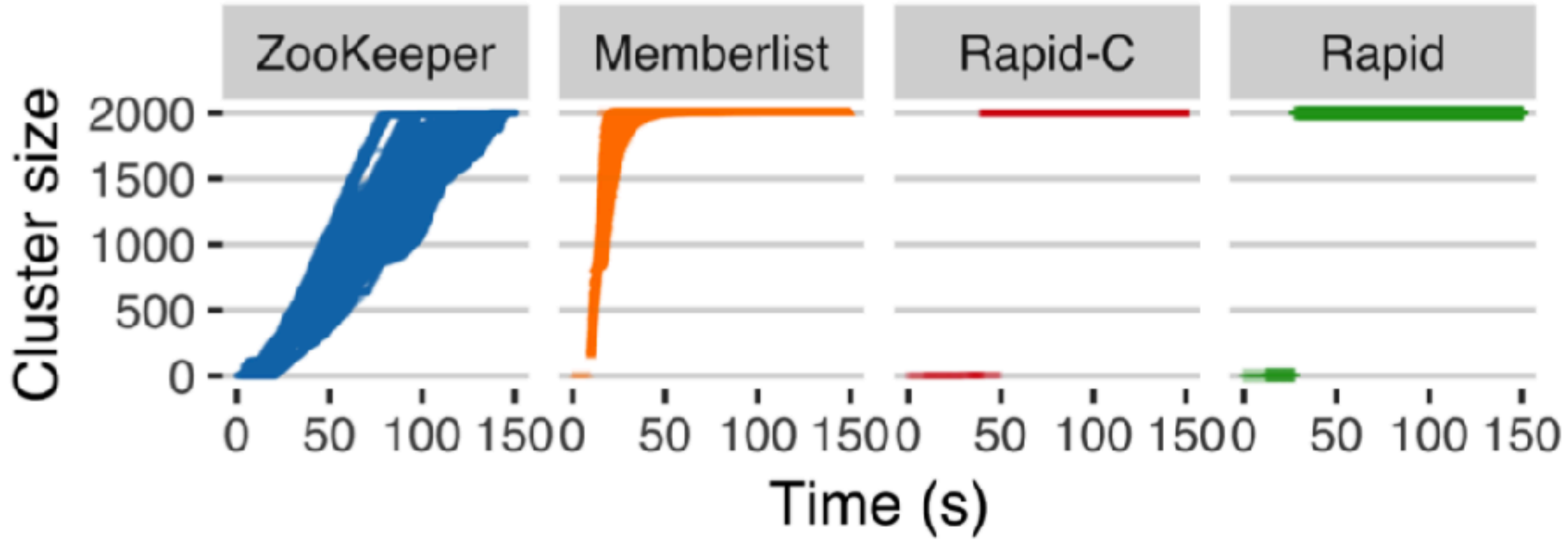


Figure 7: Timeseries showing the first 150 seconds of all three systems bootstrapping a 2000 node cluster.