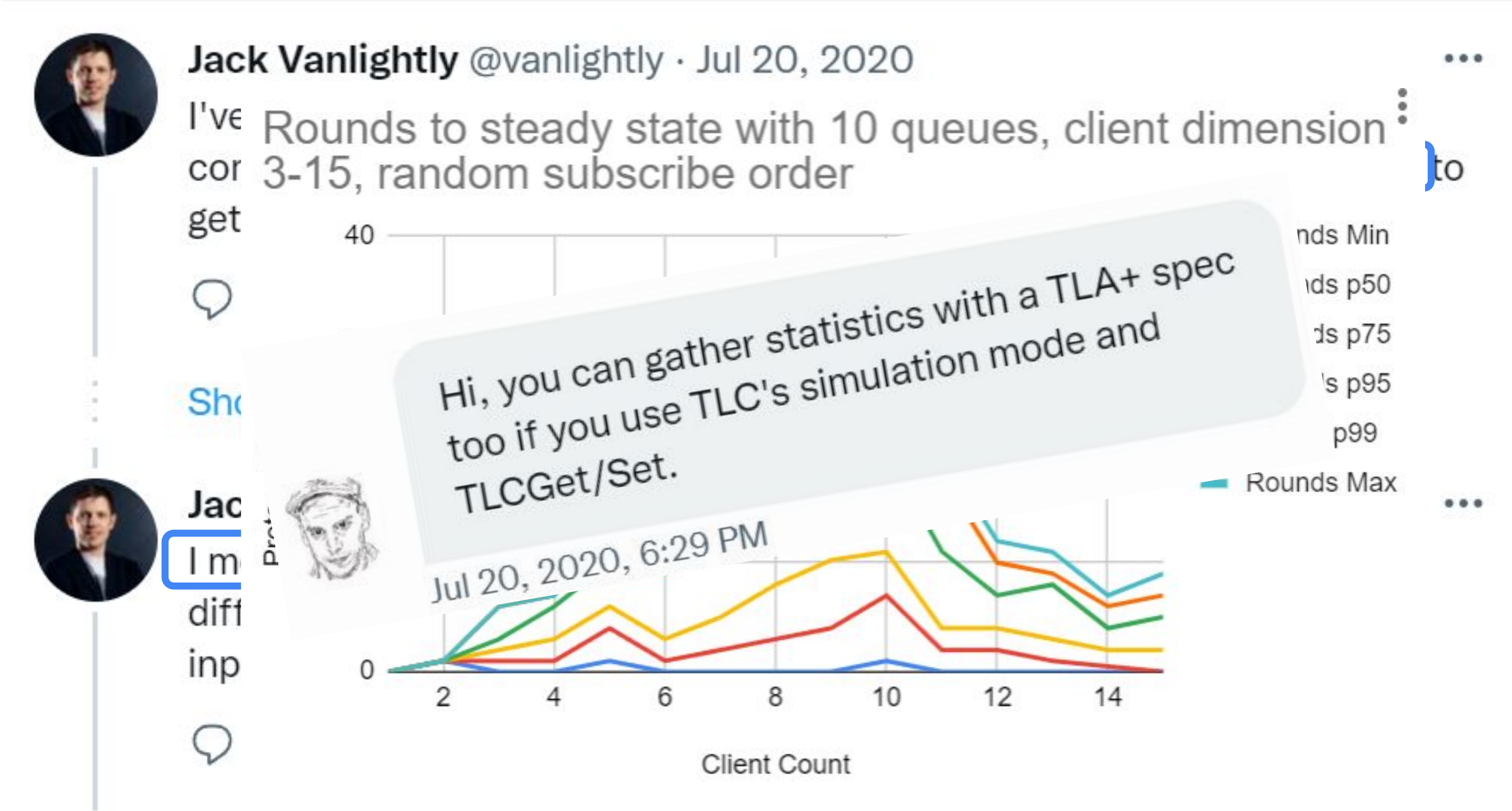
Obtaining statistical properties by simulating specs with TLC

Jack Vanlightly & Markus A. Kuppe

Jack Vanlightly
Researcher at Confluent

Markus A. Kuppe
Principal Research Engineer



Jack Vanlightly @vanlightly · Jul 20, 2020
I've
cor
get
Sho
Jac
I m
diff
inp
Rounds to steady state with 10 queues, client dimension 3-15, random subscribe order
40
0
2
4
6
8
10
12
14
Client Count
Rounds Max
Hi, you can gather statistics with a TLA+ spec too if you use TLC's simulation mode and TLCGet/Set.
Jul 20, 2020, 6:29 PM

Marc Brooker
@MarcJBrooker

This week I'm doing an internal talk at Amazon about an approach to system design that I use a lot, and think would use useful to a lot of people: simulation. This thread is a summary of the talk 1/

Marc Brooker @MarcJBrooker · Mar 16
Replying to @MarcJBrooker
Formal tools (like P and TLA+) do a great job of helping us answer questions about the safety and liveness properties of systems. Safety is obviously critical, but liveness is a bit unfulfilling. My customers don't care about 'eventually', they care about latency! 2/

1 3 62

Marc Brooker @MarcJBrooker · Mar 16
We care about cost, about throughput, about availability, about durability, etc. These are mostly probabilistic properties (there's no perfectly available system!) but we care deeply what the probabilities are ('eventually' doesn't cut it). 3/

1 2 45

Marc's Blog

Blog Posts

2022

02 Sep 2022 Histogram vs eCDF
11 Aug 2022 What is Backoff For?
29 Jul 2022 Getting into formal specification, and getting my
12 Jul 2022 The DynamoDB paper
02 Jun 2022 Formal Methods Only Solve Half My Problems
03 May 2022 What is a simple system?
11 Apr 2022 Simple Simulations for System Builders
28 Feb 2022 Fixing retries with token buckets and circuit bre
16 Feb 2022 Will circuit breakers solve my problems?
31 Jan 2022 Software Deployment, Speed, and Safety
19 Jan 2022 DynamoDB's Best Feature: Predictability

2021

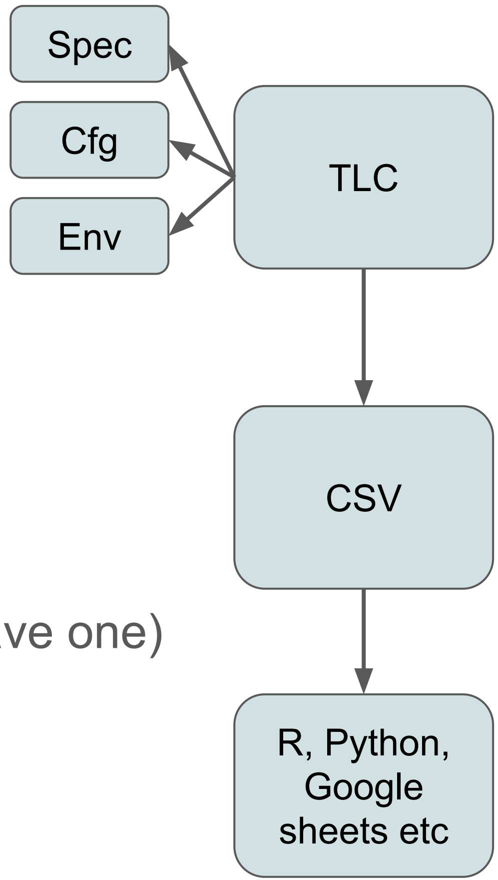
16 Nov 2021 The Bug in Paxos Made Simple
20 Oct 2021 Serial, Parallel, and Quorum Latencies

Why measure statistical properties through simulation?

- Doesn’t require engineers to have a high level of statistics
- No system-level noise with specs => Reproducible
- Evaluate hyperproperties
  - Is a property common or rare?
  - A liveness property can tell us something good eventually, but what is the distribution “time passed” across N traces?
  - Identify worst-case complexity / pathological behaviour
- Differential analysis
  - Comparing algorithm variants
  - Comparing tunable parameters
  - Seeing the impact on specification changes

How to measure statistical properties?

- Run TLC in “generation” mode
- Count events (counters)
  - Messages sent
  - Protocol rounds
- Count objects that satisfy a predicate (gauges)
  - Elements in a queue
  - Number of members that have detected a dead peer
- Cannot measure wall-clock time (use a proxy if you have one)
- Write out counters along with other attributes to CSV
- Use favourite statistical analysis tooling
  - Markus and I like R and ggplot2
- Hopefully reuse standard safety/liveness spec



Demo

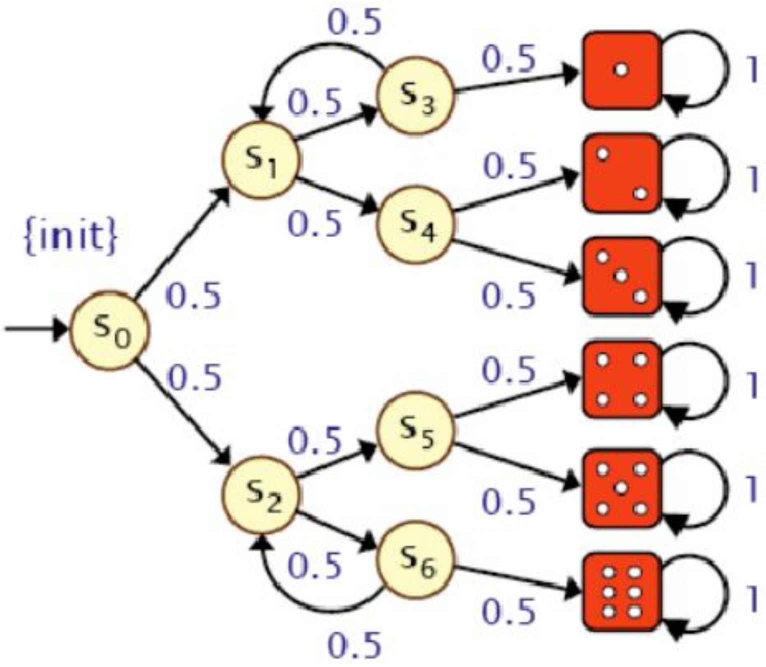
(Knuth-Yao & EWD998)

Demo

(Knuth-Yao & EWD998)

Knuth Yao - Simulate six-sided die (1976)

Knuth Yao - Simulate six-sided die (1976)



Solve Markov Chain:

- Analytically
- [Prism](), …

“[Model Checking Meets Probability: A Gentle Introduction]()“ by Joost-Pieter Katoen

Knuth Yao - Simulate six-sided die (1976)

```
----------- MODULE KnuthYao --------------
EXTENDS Reals

VARIABLES p, s, f

T == [s0 |-> [H |-> "s1", T |-> "s2"],
      s1 |-> [H |-> "s3", T |-> "s4"],
      s2 |-> [H |-> "s5", T |-> "s6"],
      s3 |-> [H |-> "s1", T |-> "1" ],
      s4 |-> [H |-> "2",  T |-> "3" ],
      s5 |-> [H |-> "4",  T |-> "5" ],
      s6 |-> [H |-> "6",  T |-> "s2"]]

Init == s = "s0" /\ p = 1 /\ f \in {"H","T"}

Next == /\ s \notin 1..6 /\ s' = T[s][f]
        /\ p' = p/2      /\ f' \in {"H","T"}
==========================================
```

A spec of a *fair* die?

- TLC no support for ℝ
- Infinite state space
  - Halving *p* in the cycles
- Cannot state “fair die” property in TLA+
  - Not true/false of a behavior
  - \A d \in 1..6: Pr(<>(s=d)) = 1/6

KnuthYao spec by Ron Pressler

Crooked Coin - Simulate six-sided die

```
$ bin/prism die.pm -ss

…

Printing steady-state probabilities in plain text format below:

7:(7,1)=0.2882349195996611

8:(7,2)=0.2882349195996611

9:(7,3)=0.12352925125699758

10:(7,4)=0.18607580157067485

11:(7,5)=0.0797467721017178

12:(7,6)=0.034177188043593335
```

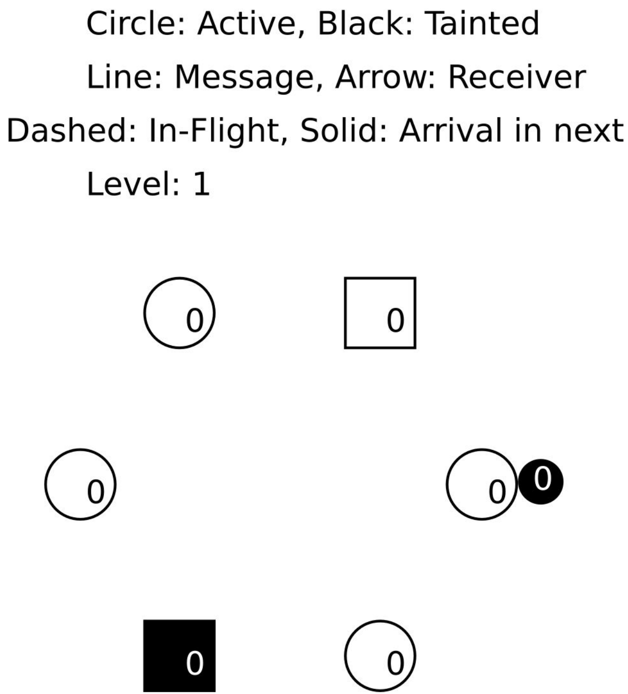
Demo

(Knuth-Yao & EWD998)

Outline Demo - Knuth Yao

1. Show graph on slides
2. Show Ron’s spec
3. Dyadic Rationals
4. Environment checks (assumes)
    a. Simulate with -depth 5
5. CSVWrite header
6. CSVWrite in “terminal” state
7. Why not all *p* \in 2^p in plot? => Some values of *p* are not *Done* states
8. Crooked Heads/Tails
    a. Stateless and Stateful

EWD998 - Termination Detection in a Ring



- An active node may send a message
- A message activates and taints the receiver node
- Initiator sends token around the (overlay) ring
- Initiator detects termination iff token
  - at initiator
  - untainted
  - sum of in-flight messages is zero

- Safe: *[](terminationDetected => terminated)*
- Live: *terminated ~> terminationDetected*

https://www.cs.utexas.edu/users/EWD/ewd09xx/EWD998.PDF

EWD998 - Proposed Optimizations (Safe & Live hold)

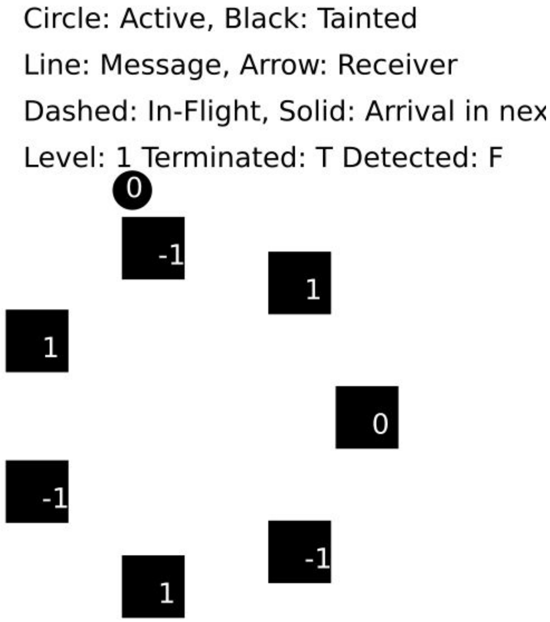
An active node may pass the token if the node is black.

```
PassToken(i) ==
  (* Rules 2 + 4 + 7 *)
  /\ ~ active[i] \* If machine i is active,
  /\ \/ ~ active[i] \* If machine i is acti
     \/ color[i] = "black"
  /\ token.pos = i
  /\ token' = [token EXCEPT !.pos = @ - 1,
```

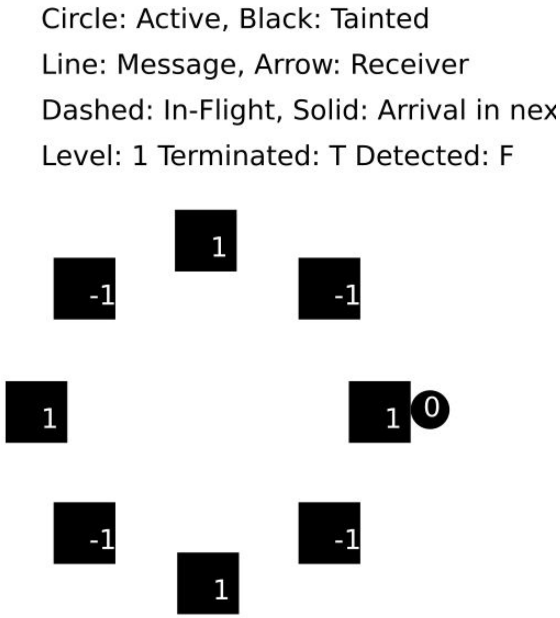
```
PassToken(i) ==
  (* Rules 2 + 4 + 7 *)
  /\ ~ active[i] \* If machine i is active, keep the token.
  /\ token.pos = i
  /\ token' = [token EXCEPT !.pos = @ - 1,
  /\ token' = [token EXCEPT !.pos = IF color[i] = "black" THEN 0 ELSE @ - 1,
                            !.q = @ + counter[i],
```

An node returns the token to the initiator if the node is black, i.e., abort inconclusive token round.

EWD998 - Deoptimization Analyzed



Token passes:
O(3N)



Token passes:
O(((N+1)² + (N+1) / 2) - 1)

Outline Demo - EWD998

1. Intro termination detection algorithm
2. Outline the proposed optimizations
3. Spec: Feature flags in *PassTokenOpts*
4. Spec: Hooks in *AtTermination* and *AtTerminationDetection*
5. Spec: Validation with asserts in *AtTerminationDetection*
6. Spec: Decreasing probability of *SendMsgOpts*
7. Spec: “Script” *EWD998_optsSC.tla* and *IOUtils!IOEnv*
8. Graph: T2TD
    a. Point out that all optimization combinations are simulated
9. Graph: occurrence of actions

Case Studies

(SWIM, RabbitMQ, Kafka)

SWIM

https://www.cs.cornell.edu/projects/Quicksilver/public_pdfs/SWIM.pdf

- Group membership
- Failure detection component
- Dissemination component - infection style

HashiCorp

Serf

Decentralized Cluster Membership, Failure Detection, and Orchestration.

SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol

Abhinandan Das, Indranil Gupta, Ashish Motivala*
Dept. of Computer Science, Cornell University
Ithaca NY 14853 USA
{asdas,gupta,ashish}@cs.cornell.edu

Abstract

Several distributed peer-to-peer applications require weakly-consistent knowledge of process group membership information at all participating processes. SWIM is a generic software module that offers this service for large-scale process groups. The SWIM effort is motivated by the unscalability of traditional heart-beating protocols, which either impose network loads that grow quadratically with group size, or compromise response times or false positive frequency w.r.t. detecting process crashes. This paper reports on the design, implementation and performance of the SWIM sub-system on a large cluster of commodity PCs.

Unlike traditional heartbeating protocols, SWIM separates the failure detection and membership update dissemination functionalities of the membership protocol. Processes are monitored through an efficient peer-to-peer periodic randomized probing protocol. Both the expected time to first detection of each process failure, and the expected message load per member, do not vary with group size. Information about membership changes, such as process joins, drop-outs and failures, is propagated via piggybacking on ping messages and acknowledgments. This results in a robust and fast infection style (also epidemic or gossip-style) of dissemination.

The rate of false failure detections in the SWIM system is reduced by modifying the protocol to allow group mem-

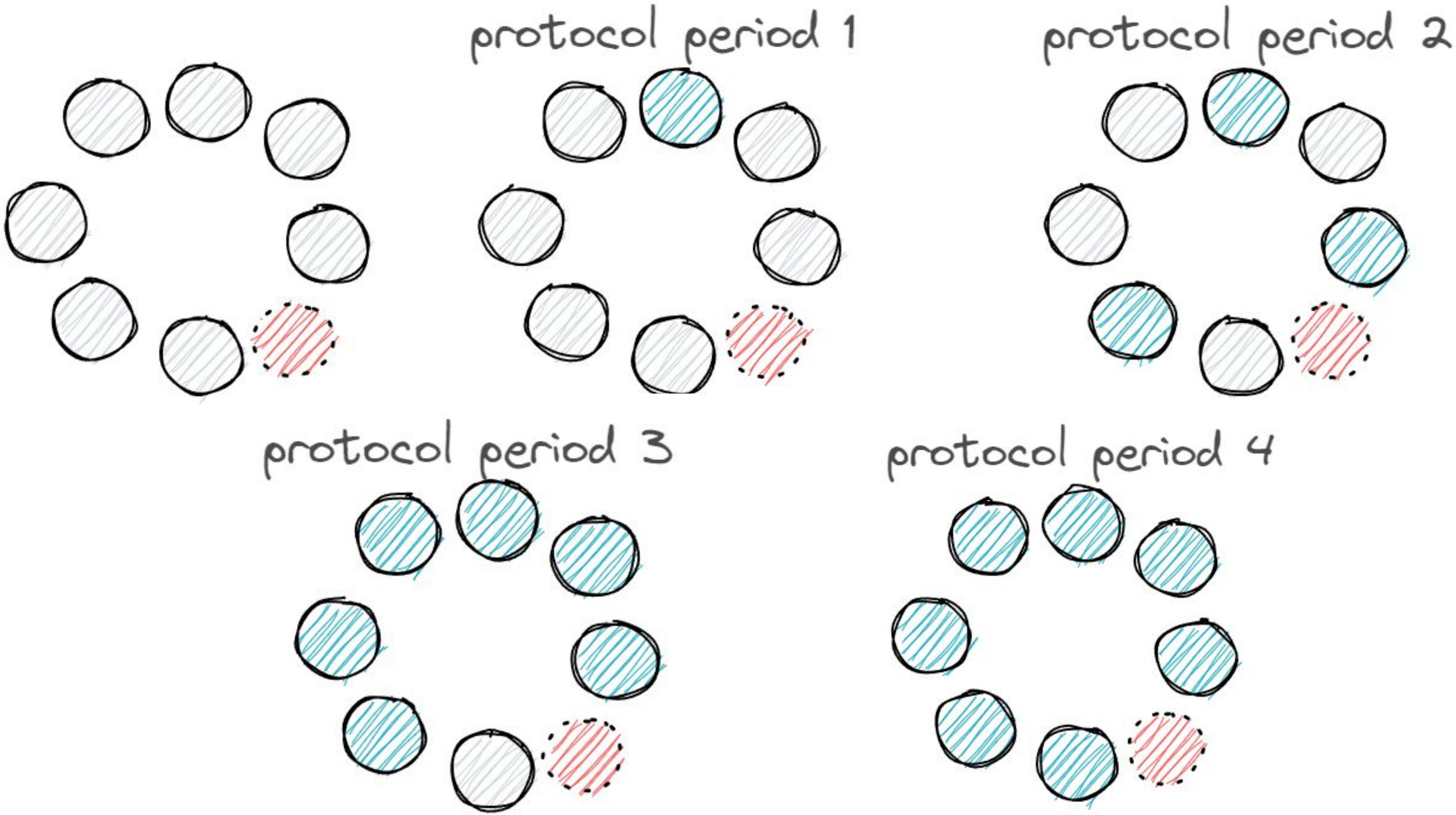
1. Introduction

As you swim lazily through the milieu,
The secrets of the world will infect you.

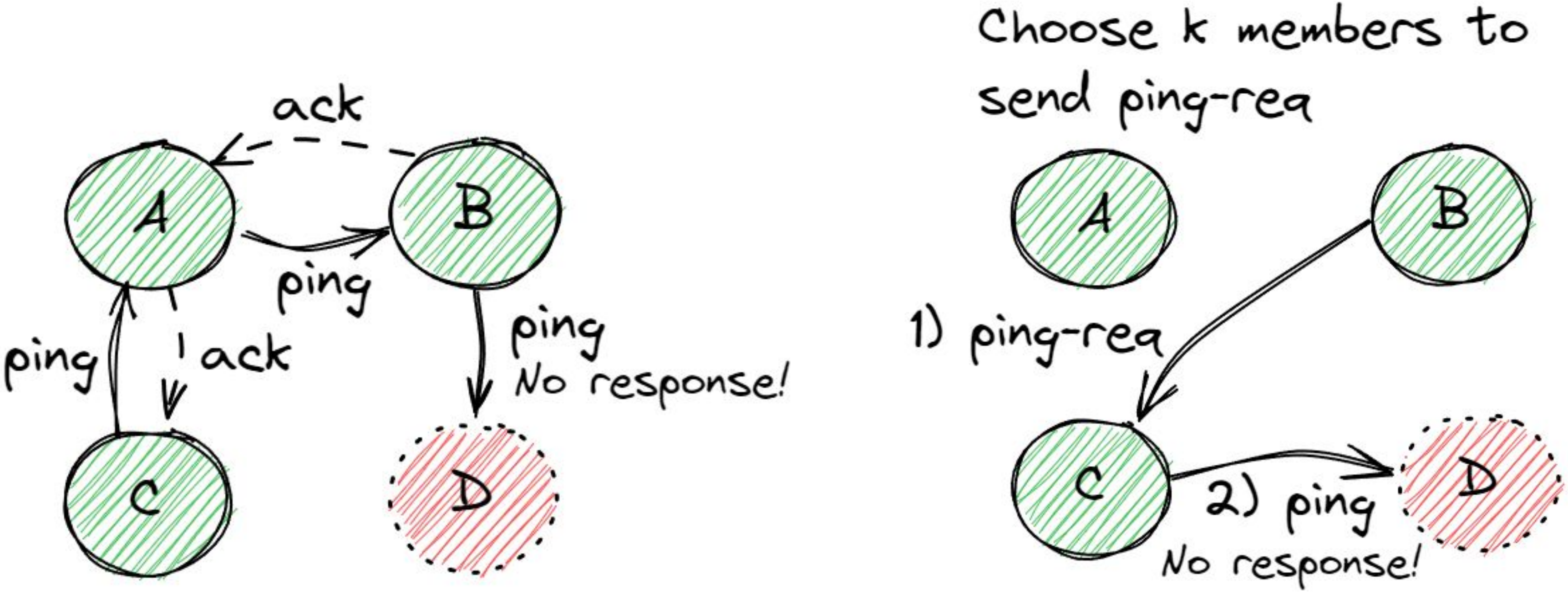
Several large-scale peer-to-peer distributed process groups running over the Internet rely on a distributed membership maintenance sub-system. Examples of existing middleware systems that utilize a membership protocol include reliable multicast [3, 11], and epidemic-style information dissemination [4, 8, 13]. These protocols in turn find use in applications such as distributed databases that need to reconcile recent disconnected updates [14], publish-subscribe systems, and large-scale peer-to-peer systems[15]. The performance of other emerging applications such as large-scale cooperative gaming, and other collaborative distributed applications, depends critically on the reliability and scalability of the membership maintenance protocol used within.

Briefly, a membership protocol provides each process ("member") of the group with a locally-maintained list of other non-faulty processes in the group. The protocol ensures that the membership list is updated with changes resulting from new members joining the group, or dropping out (either voluntarily or through a failure). The membership list is made available to the application either directly in its address space, or through a callback interface or an API. The application is free to use the contents of the list as required, e.g. gossip-based dissemination protocols would

Infection-style spread of member state information

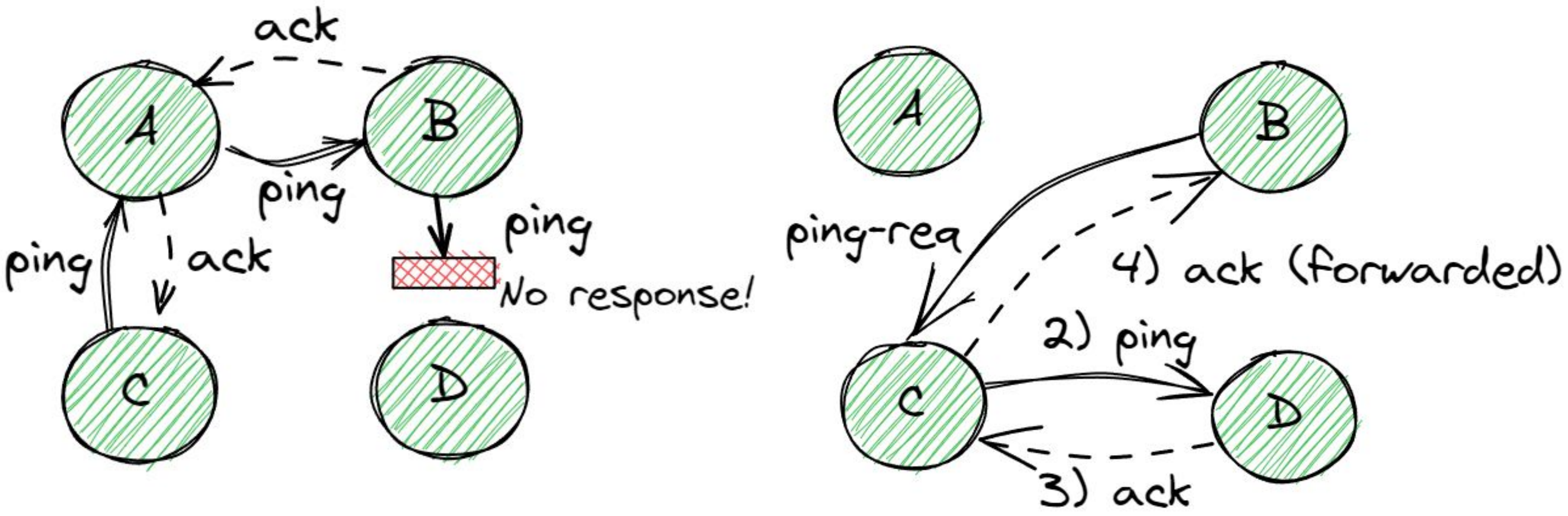


Failure detection component

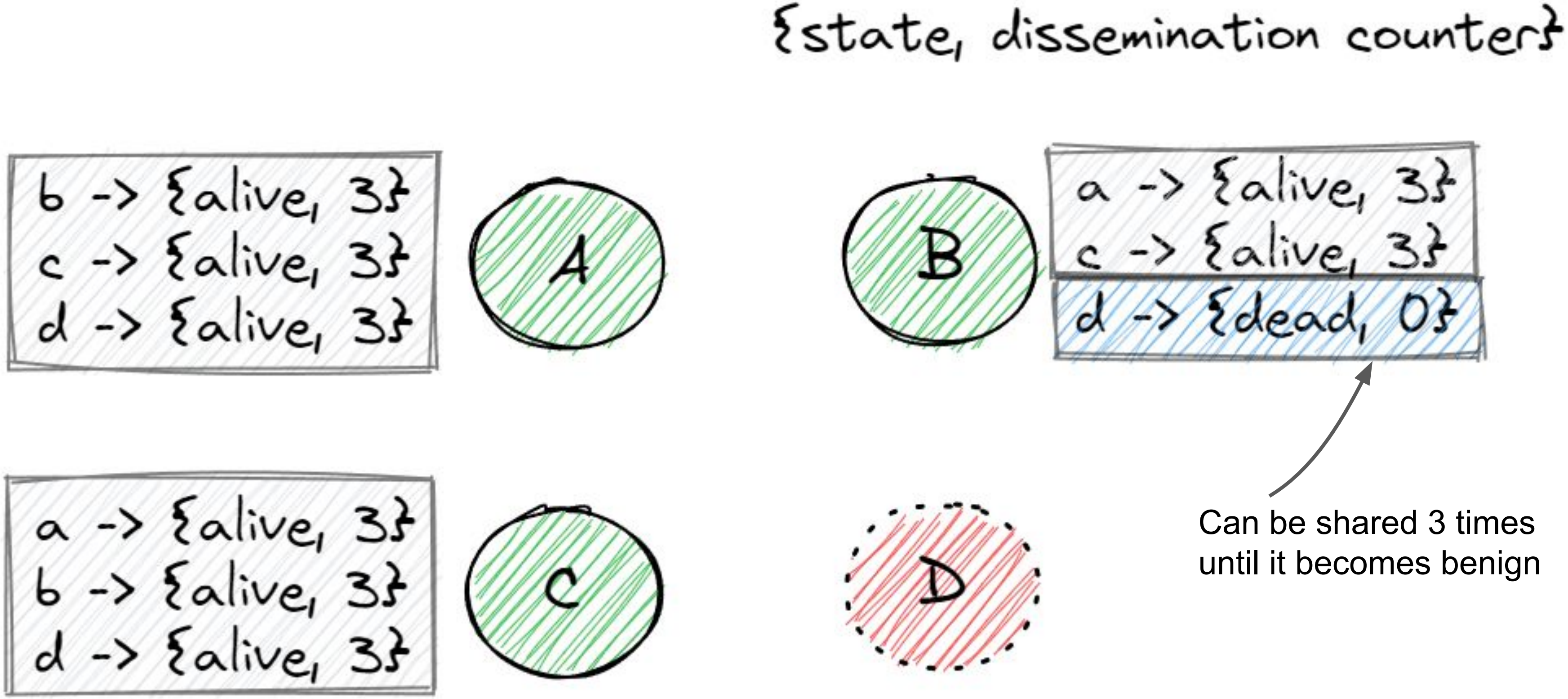


k = Peer Group Size

Failure detection component

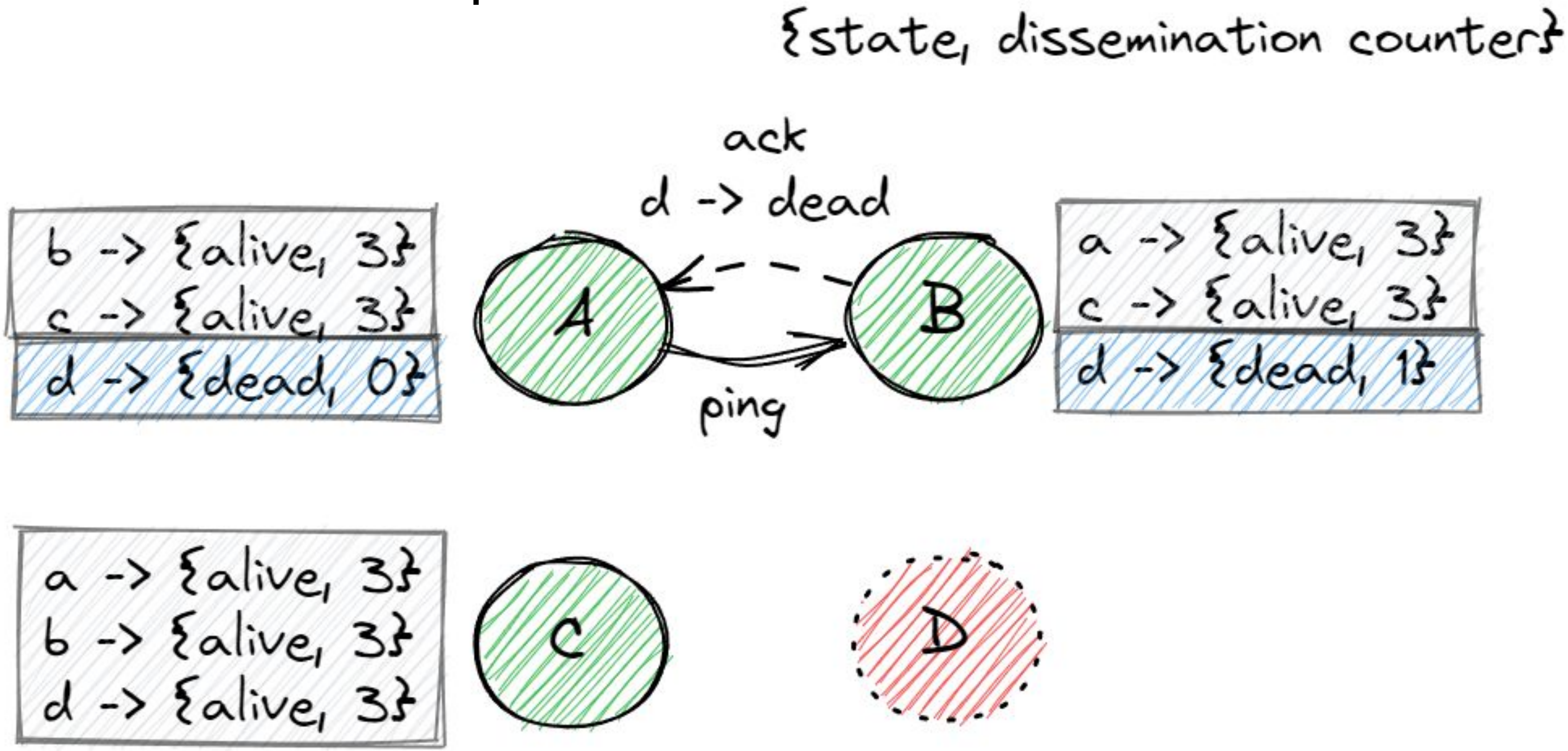


Dissemination component - information is infectious!

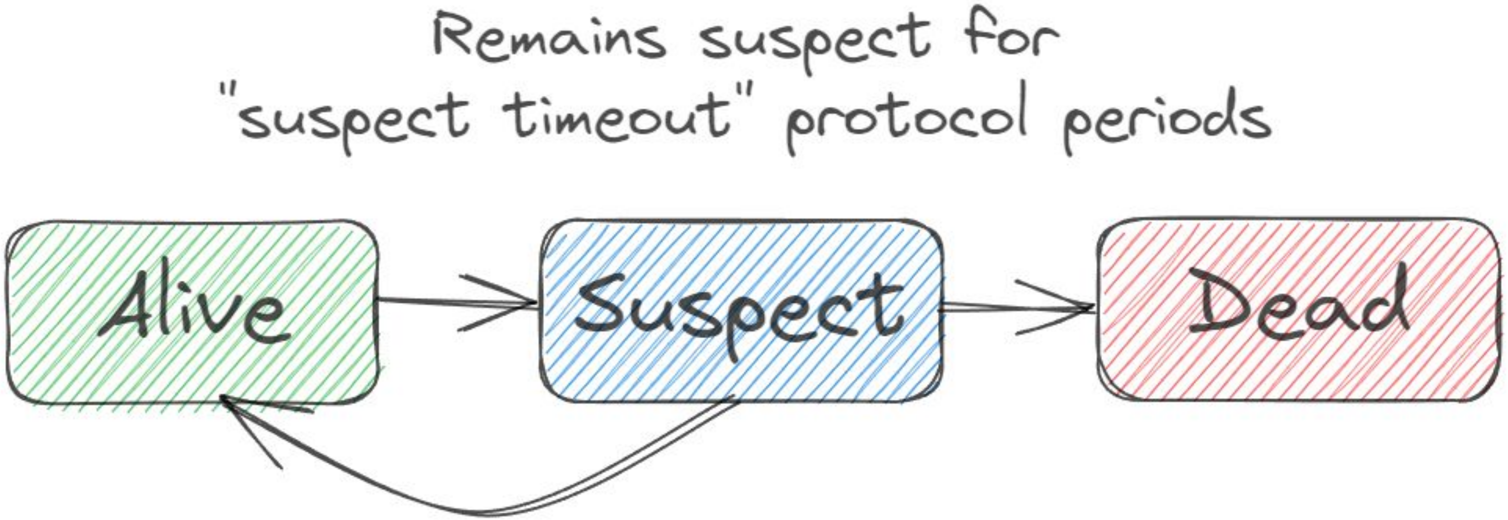


Can be shared 3 times
until it becomes benign

Dissemination component

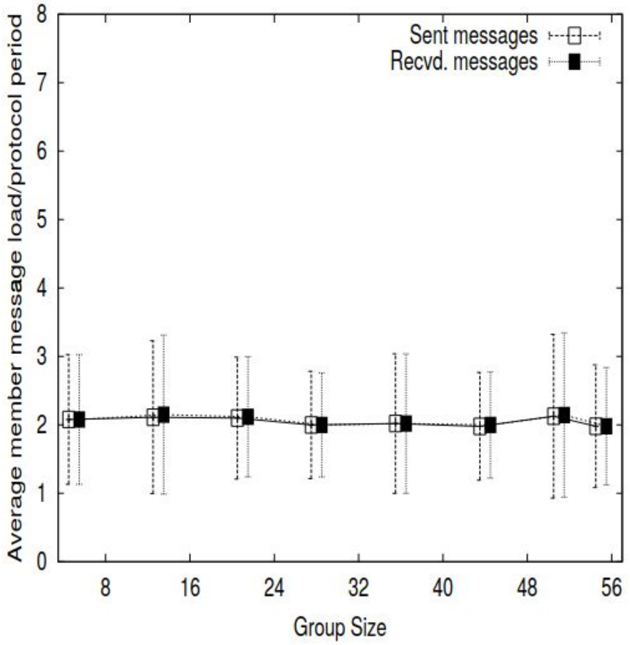


Suspicion mechanism



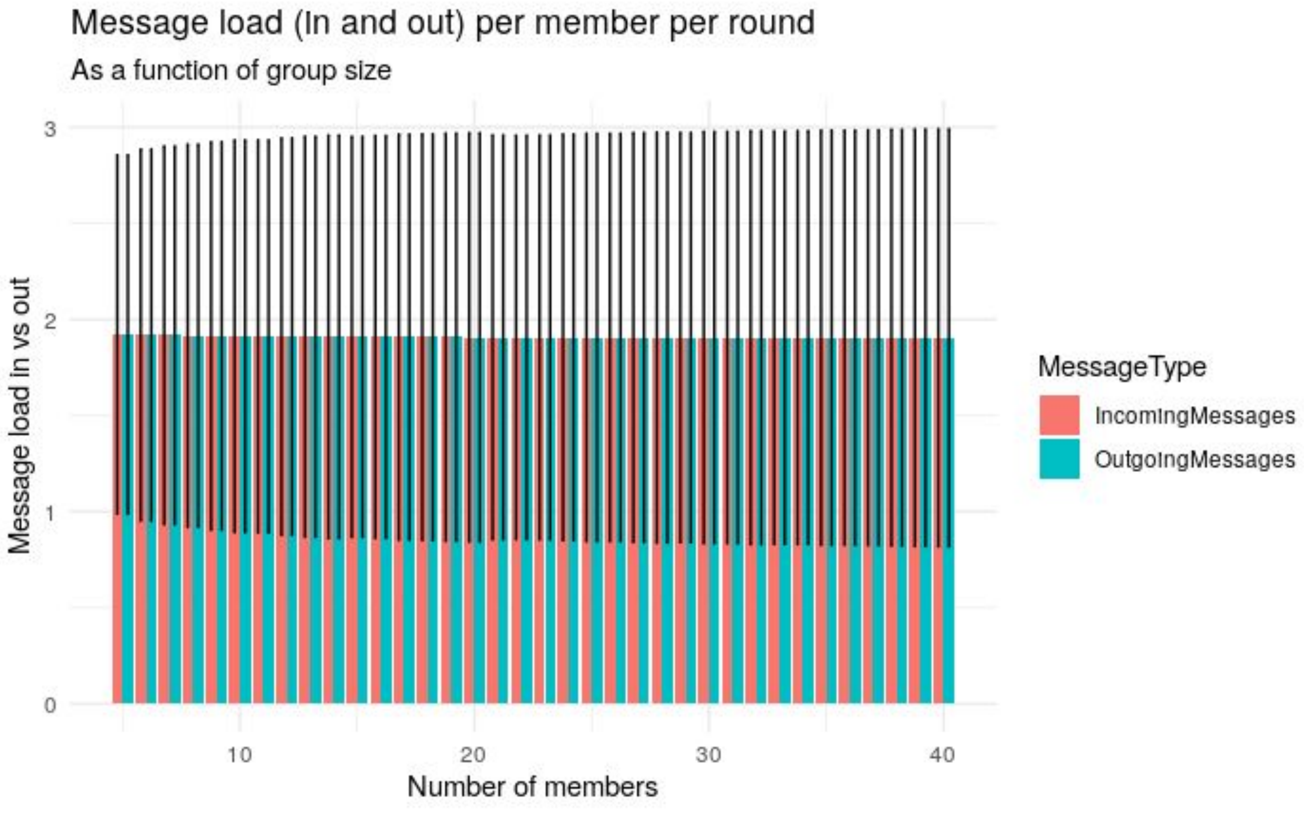
SWIM - Checking results against the paper

Message load per member per round



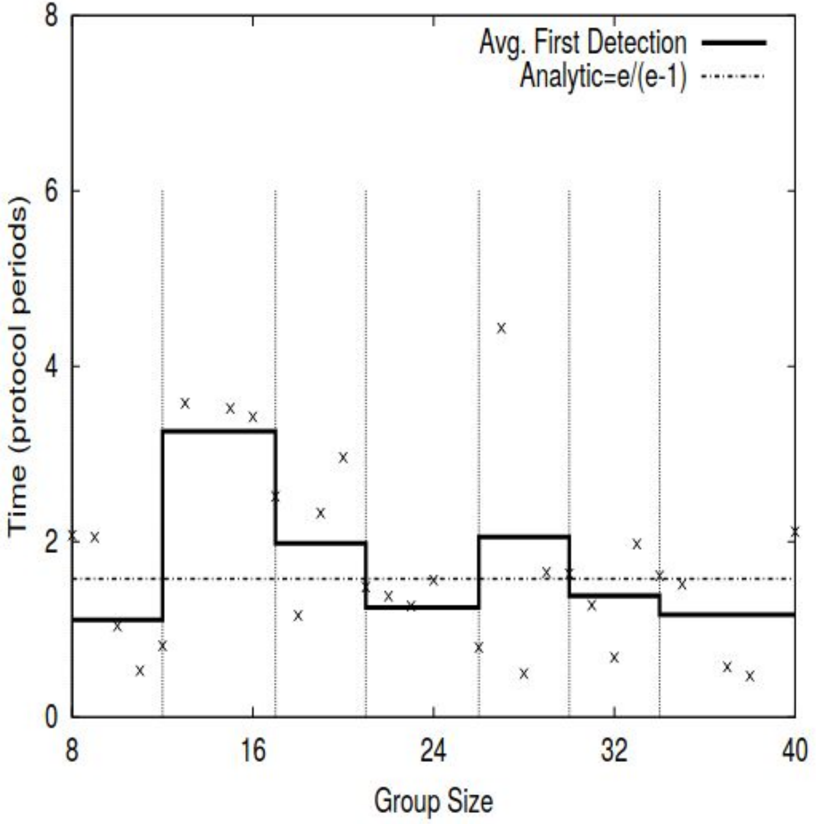
SWIM - Checking results against the paper

Message load per member per round



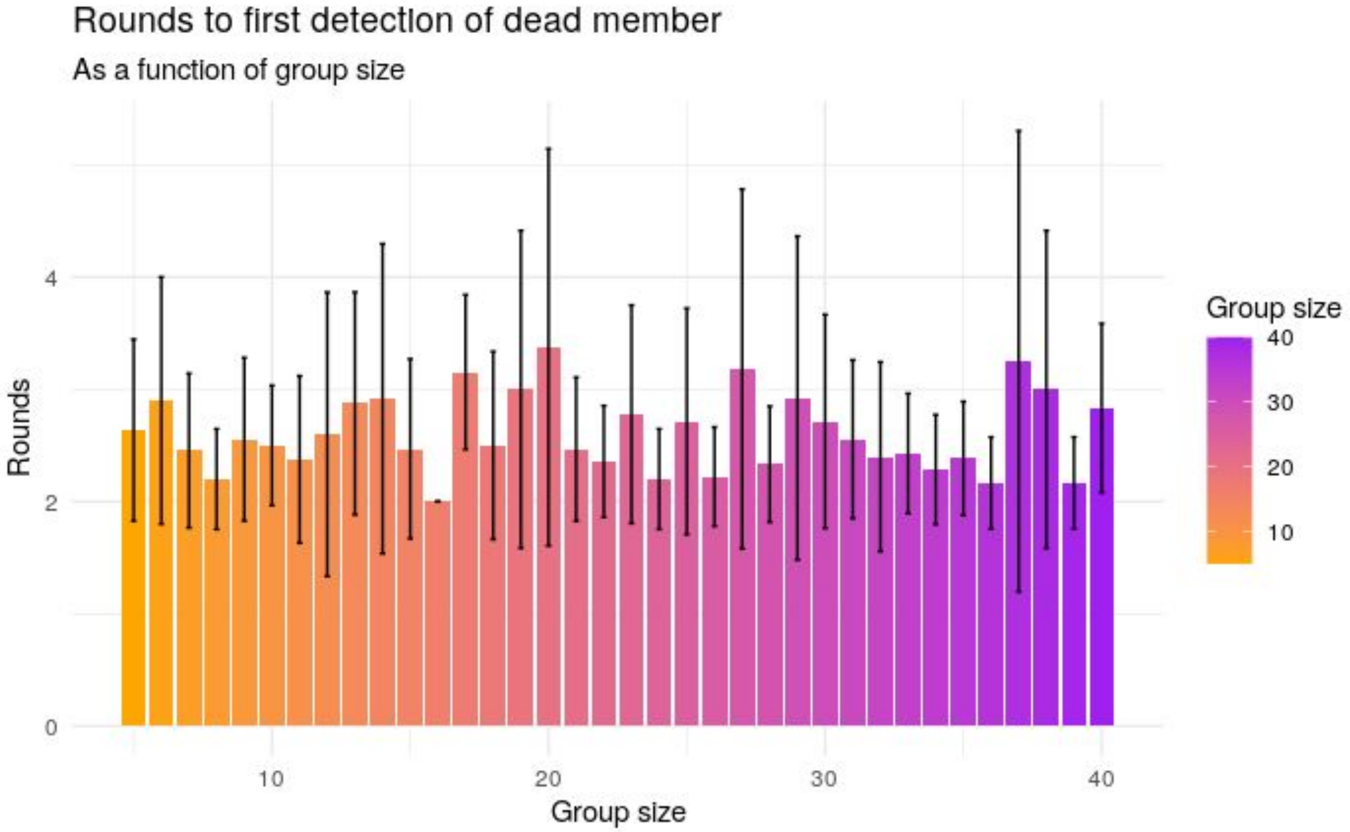
SWIM - Checking results against the paper

Protocol rounds to first detection of dead member



SWIM - Checking results against the paper

Protocol rounds to first detection of dead member

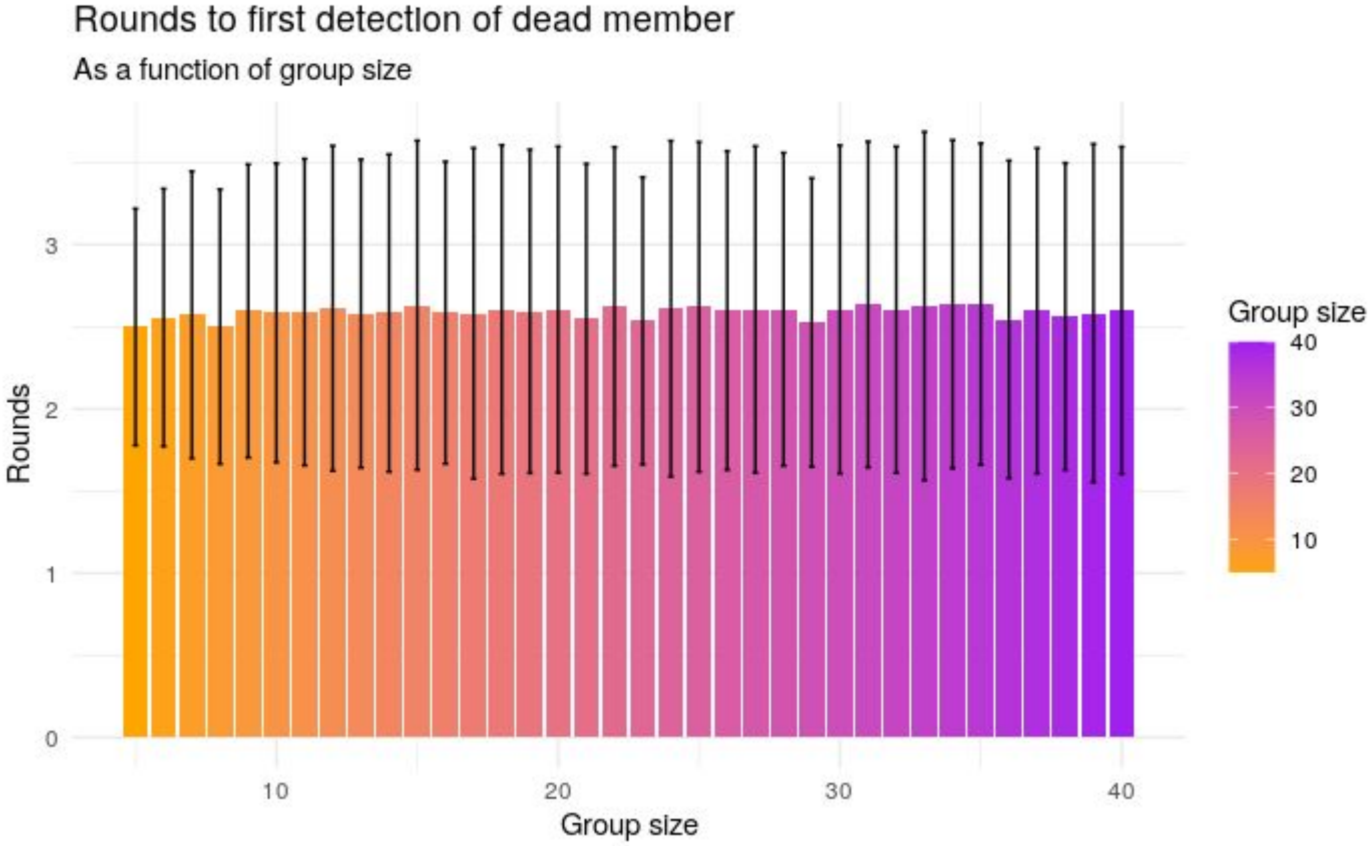


Run 10 times per group size

SWIM - Checking results against the paper

Protocol rounds to first detection of dead member

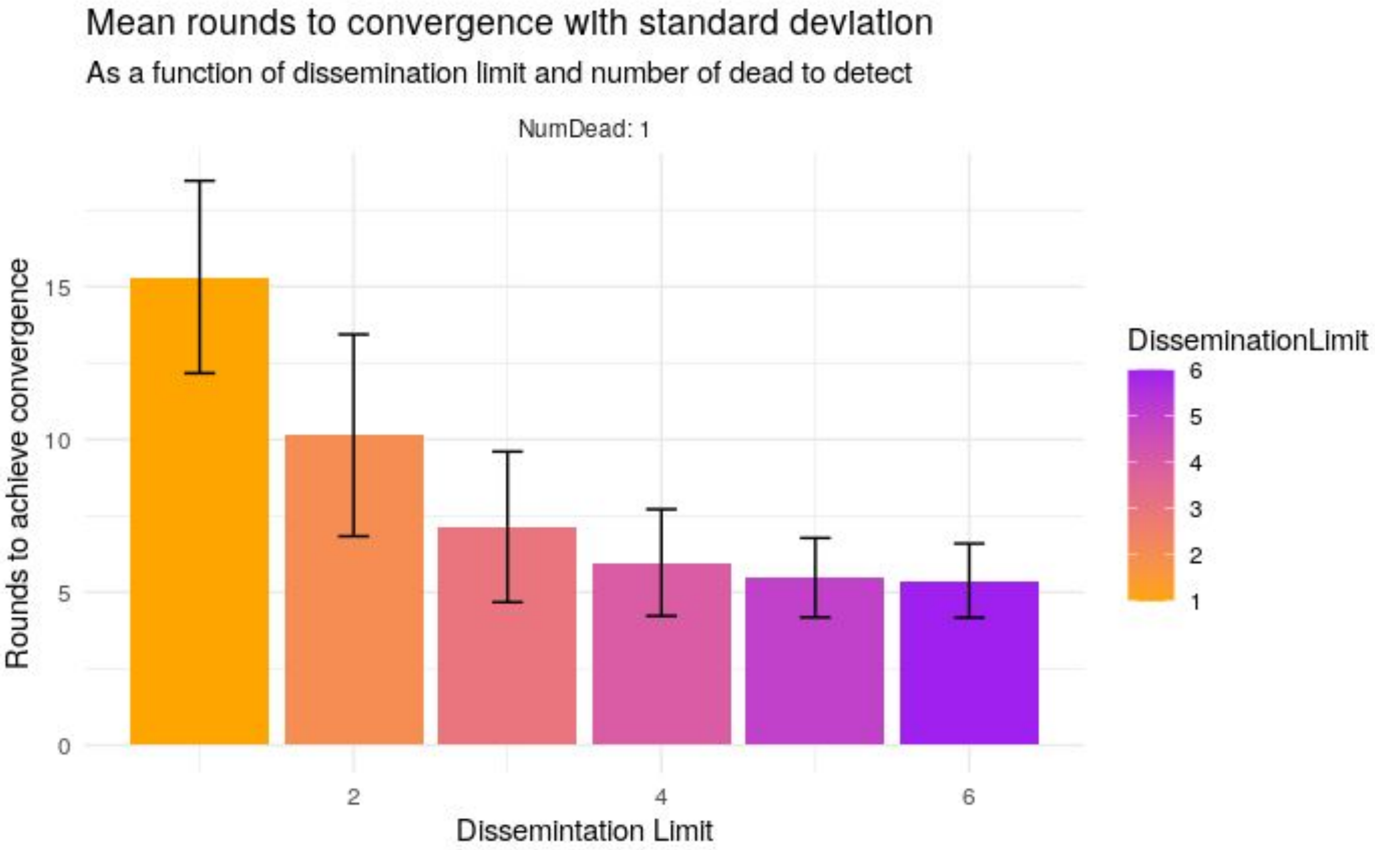
Run 1000 times per group size



Differential analysis

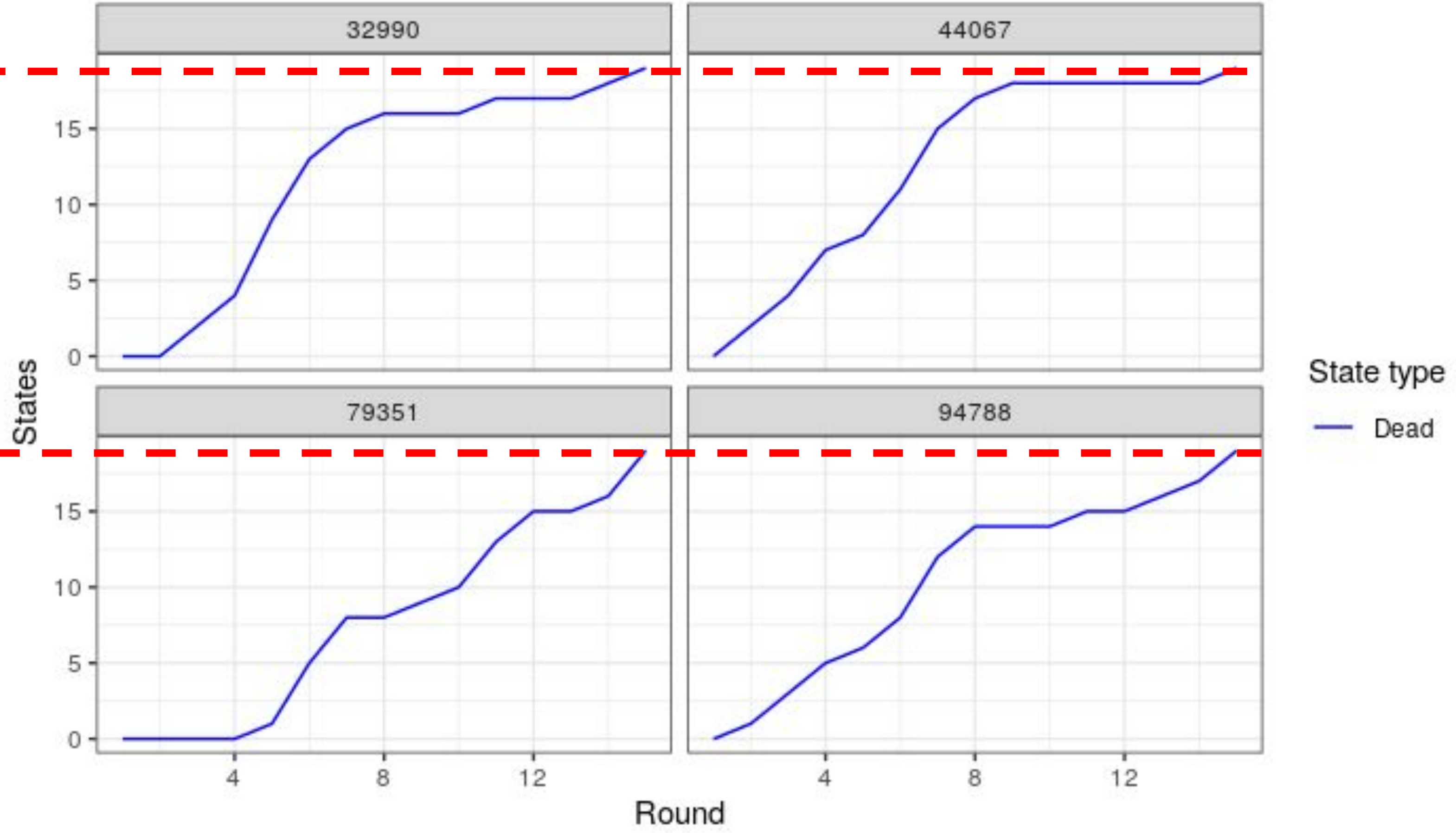
- Exploring tunable parameters as dimensions
    - Comparing algorithm variants

Dimension: Dissemination limit with group size=20



Inspecting specific traces - propagation of dead states

All members have been infected



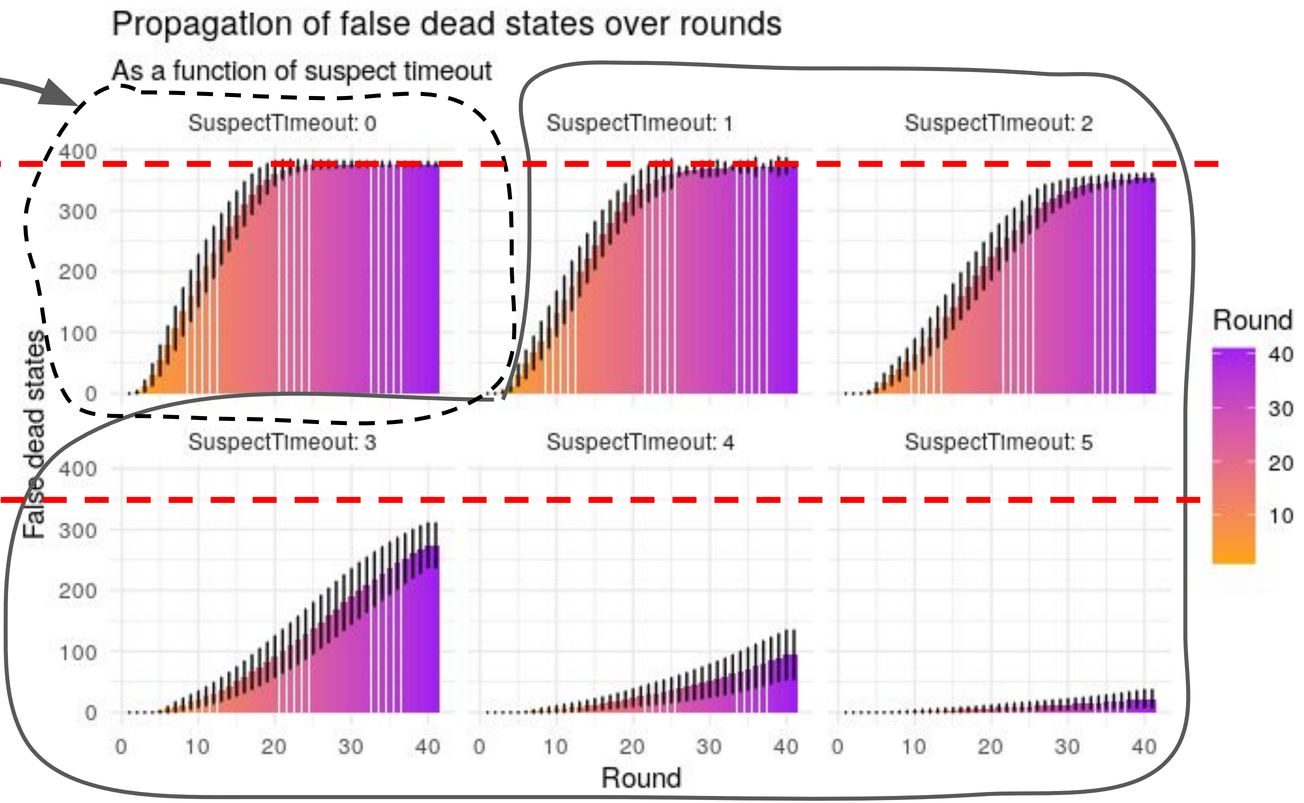
Variants: “SWIM + Inf” vs “SWIM + Inf + Susp”

Dimensions: Suspect Timeout (with 10% message loss)

SWIM + Inf

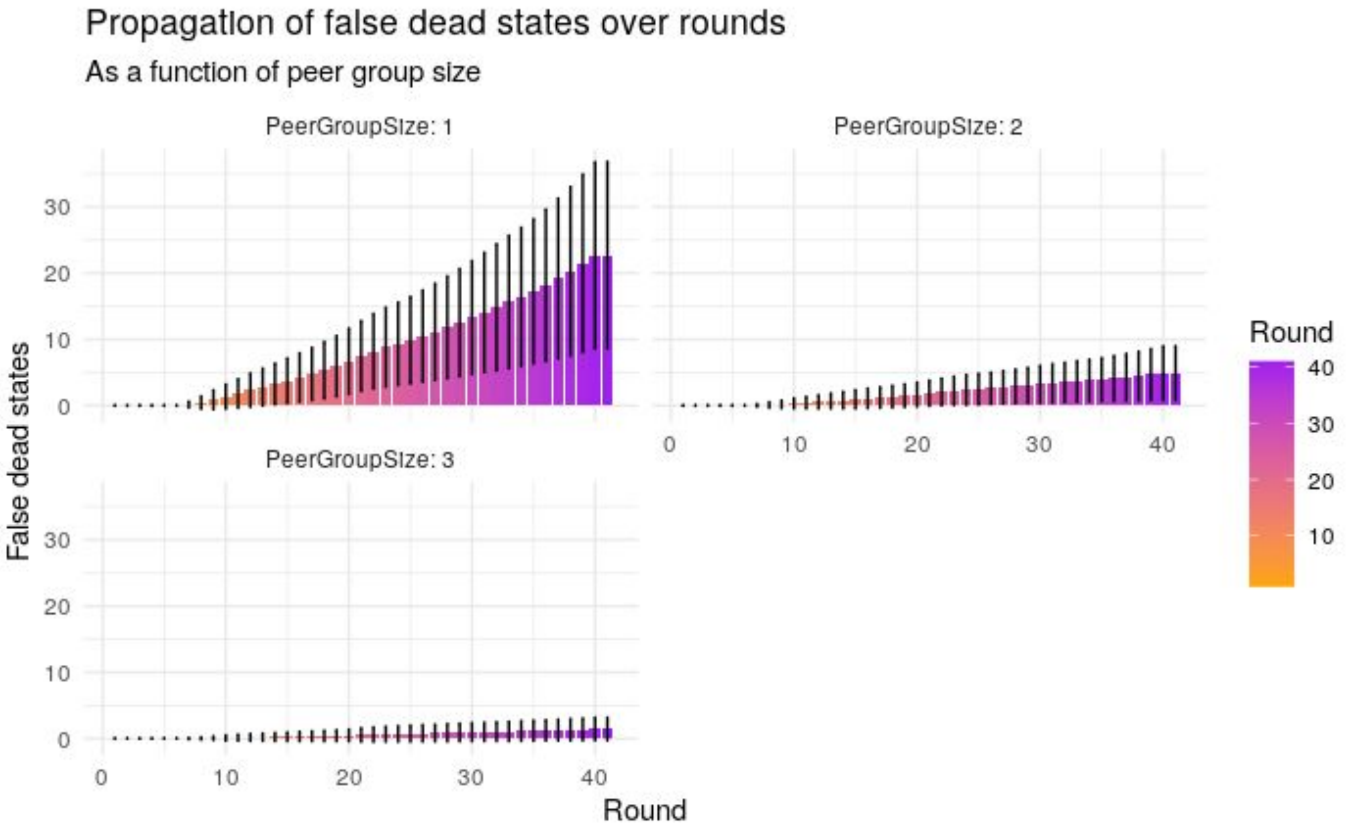
All members think all other member are deadº

SWIM + Inf + Susp

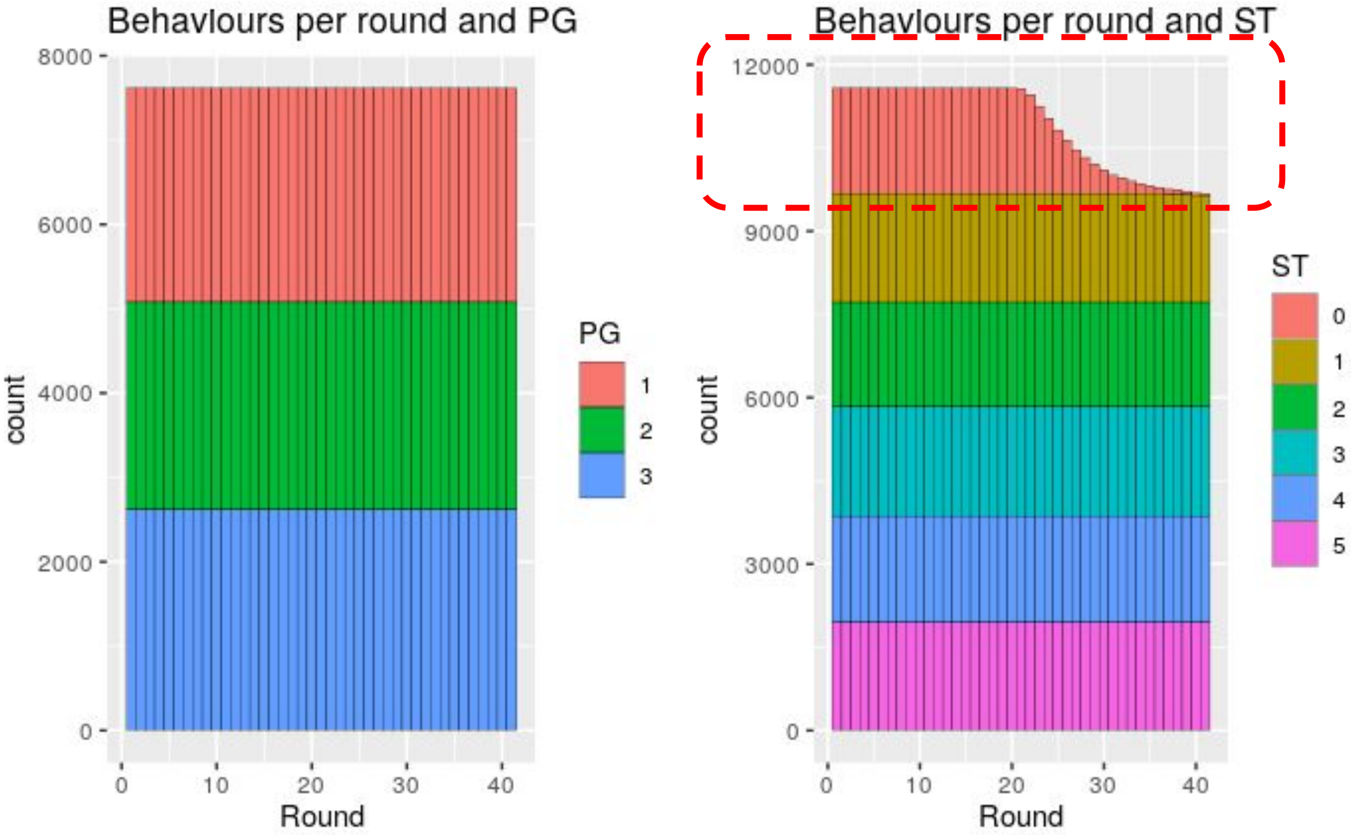


Variant: “SWIM + Inf + Susp” and Suspect Timeout=5

Dimension: Peer Group Size (with 10% message loss)

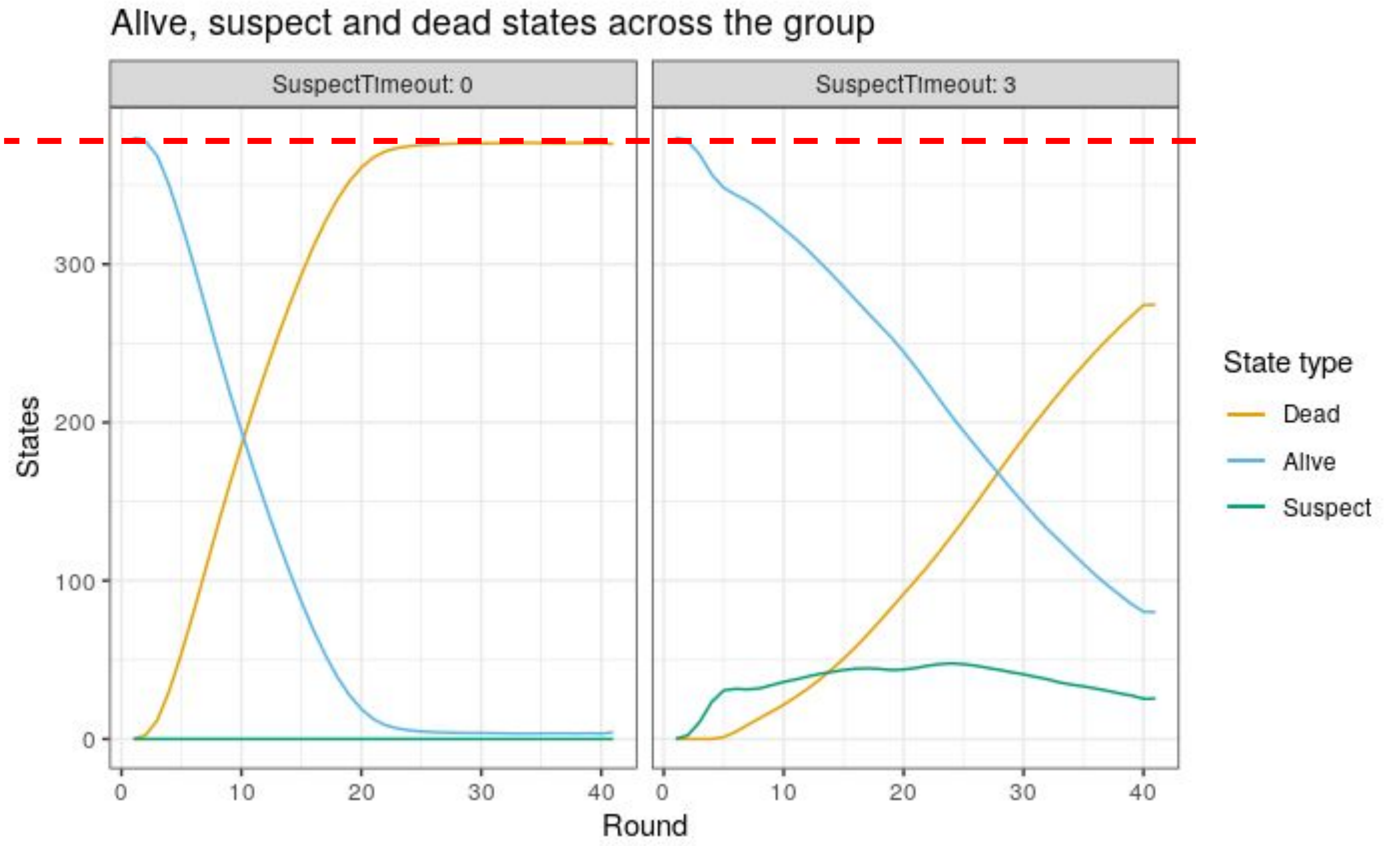


SWIM - Ensuring uniform distribution of simulation dimensions



SWIM - Ensuring uniform distribution of simulation dimensions

All members believe all other members are dead



SWIM - Challenges

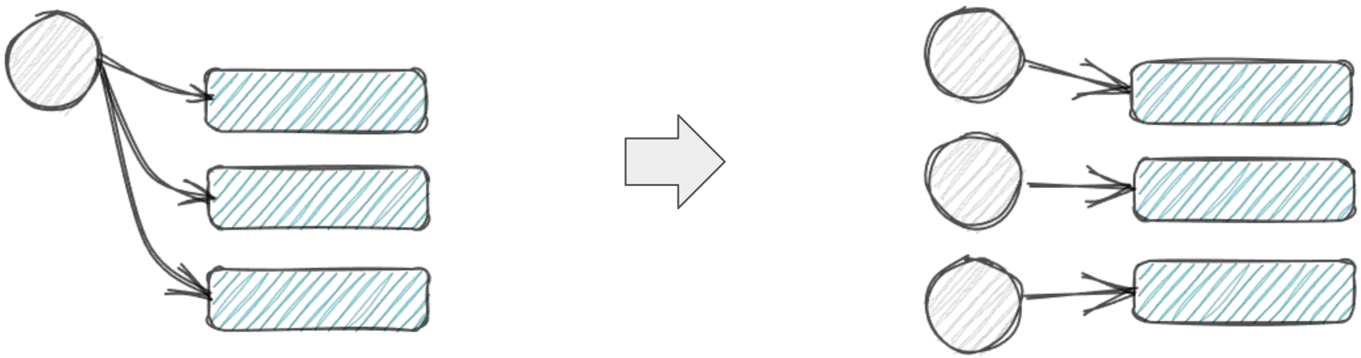
- Performance
  - Overrides required for larger models to achieve higher behaviour counts
  - TLC improvements
- Specification complexity
  - Not just basic abstraction
  - Implement variants faithfully
  - Support for configuring variants and dimensions
  - Ensuring metrics emitted at the right time

Case Studies

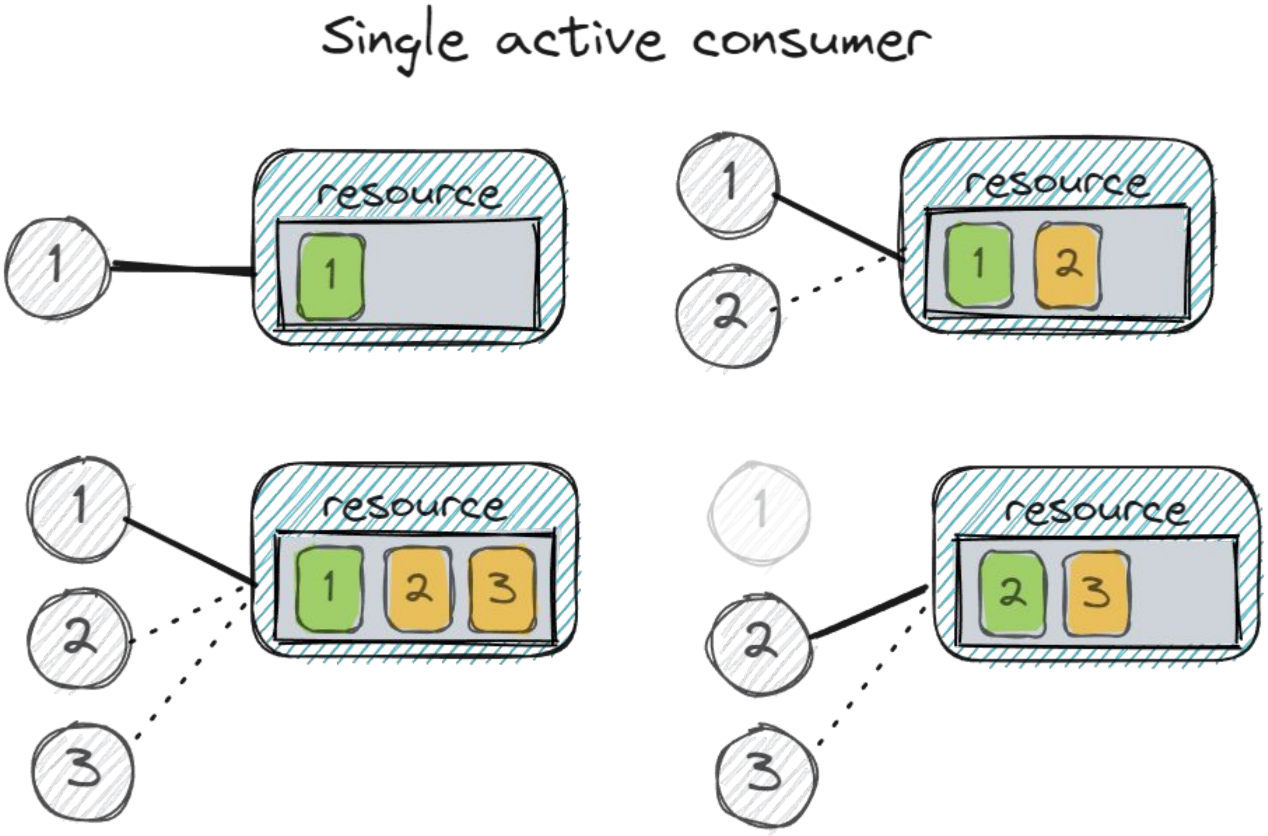
(SWIM, RabbitMQ, Kafka)

RabbitMQ Reactor Streams library

- Cooperative resource allocation
  - Multiple clients cooperate to balance queue assignment fairly
- High degree of non-determinism

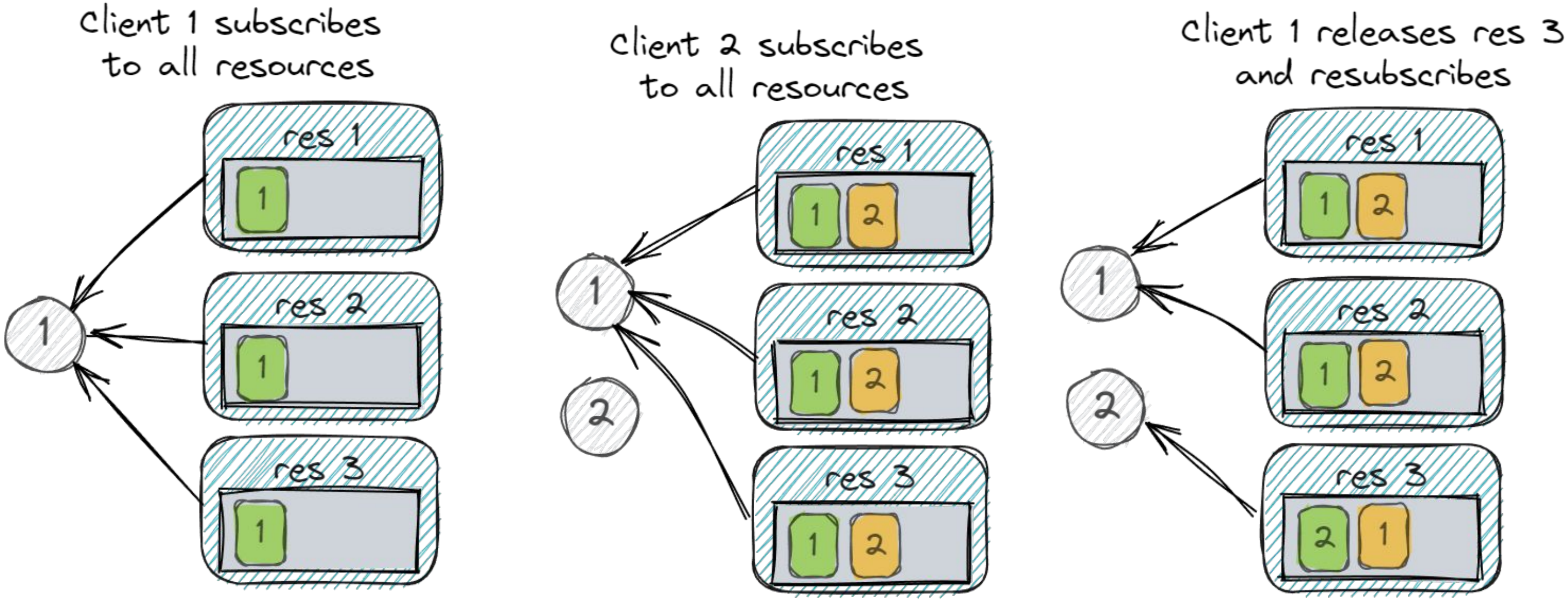


RabbitMQ Reactor Streams library

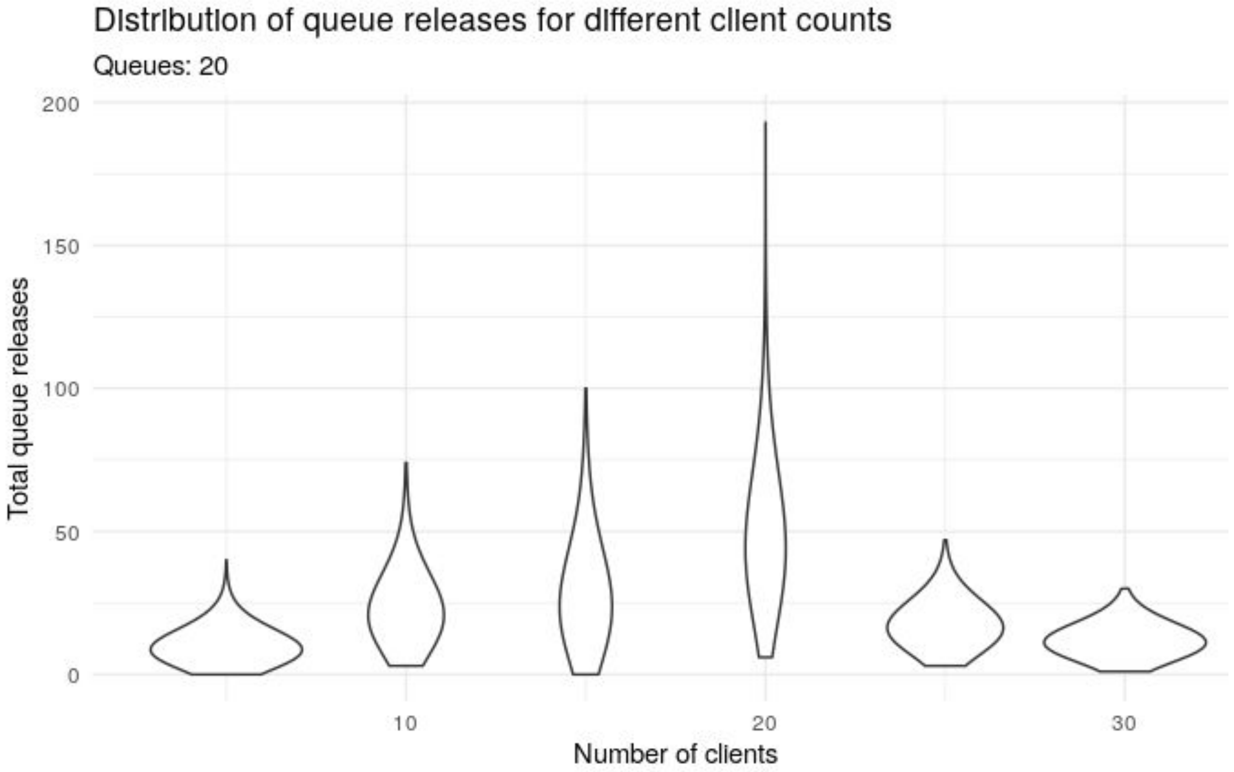


RabbitMQ Reactor Streams library

Cooperative clients



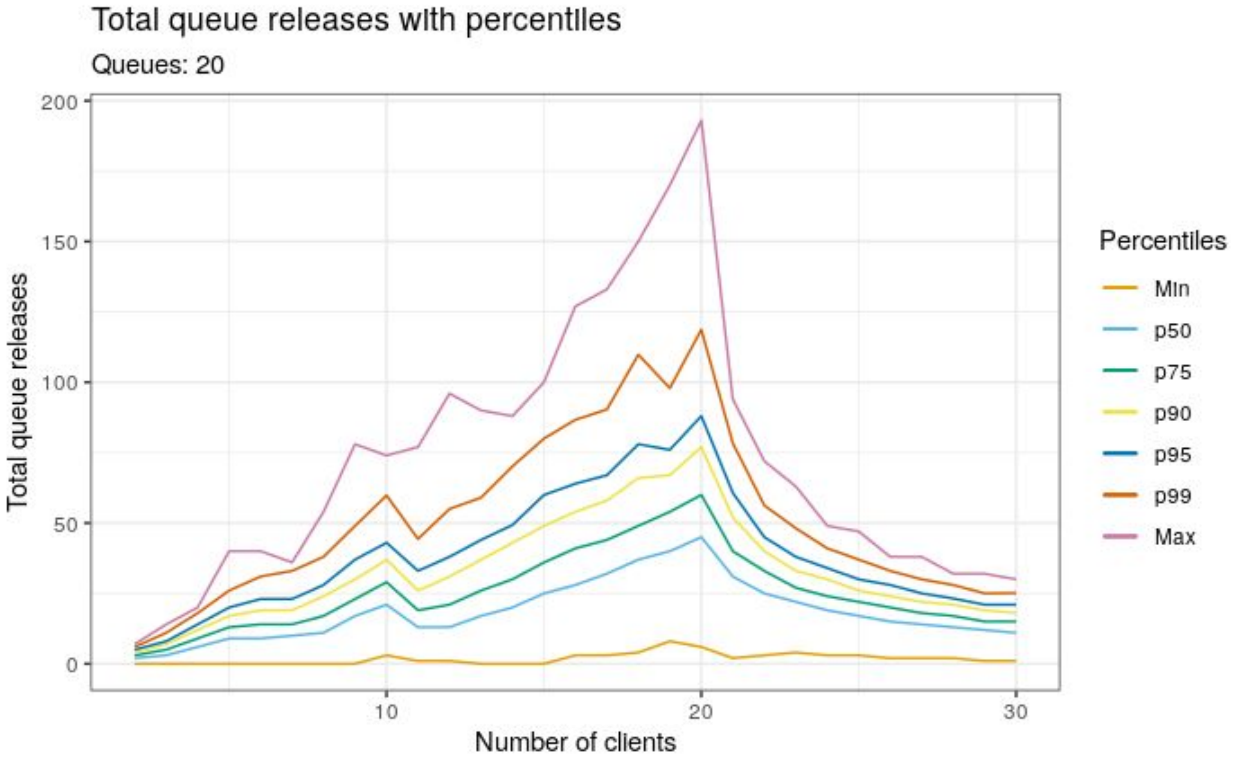
Scenario: Concurrent start-up of clients
Dimension: Number of clients
Measured: Total queue releases



Scenario: Concurrent start-up of clients

Dimension: Number of clients

Measured: Total queue releases

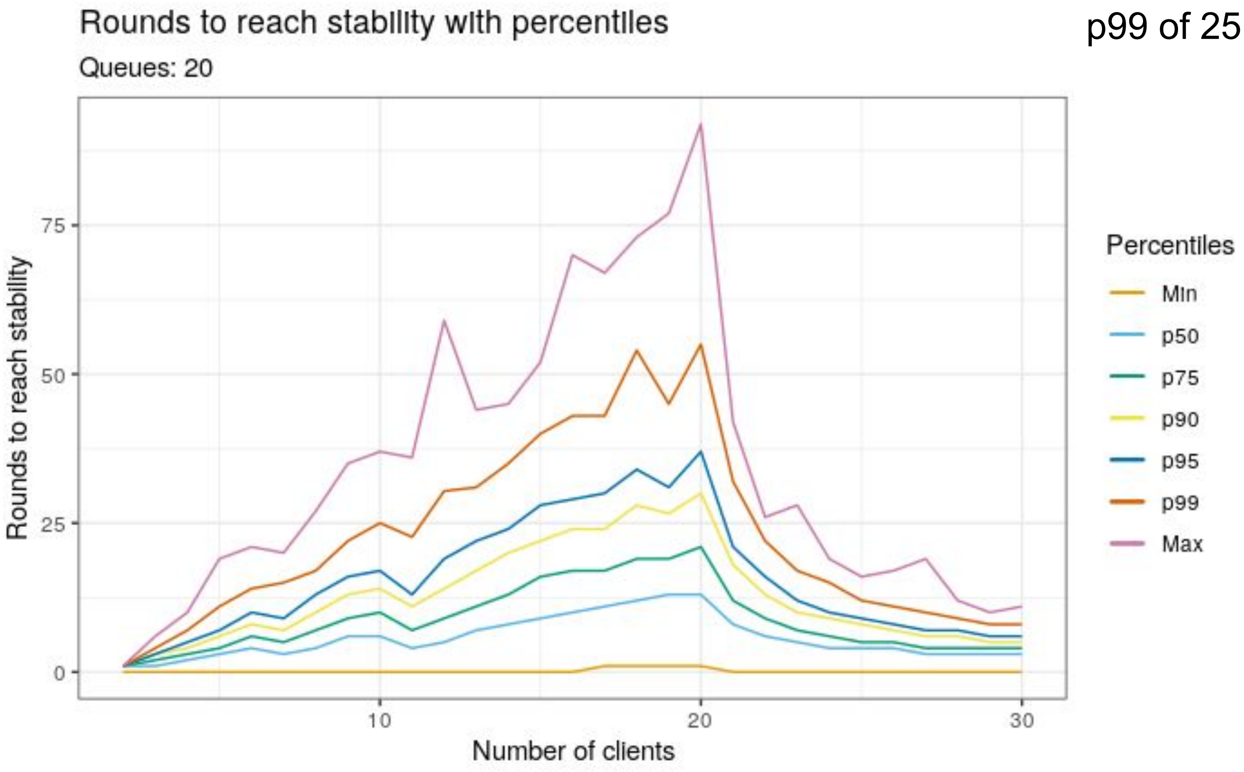


Scenario: Concurrent start-up of clients

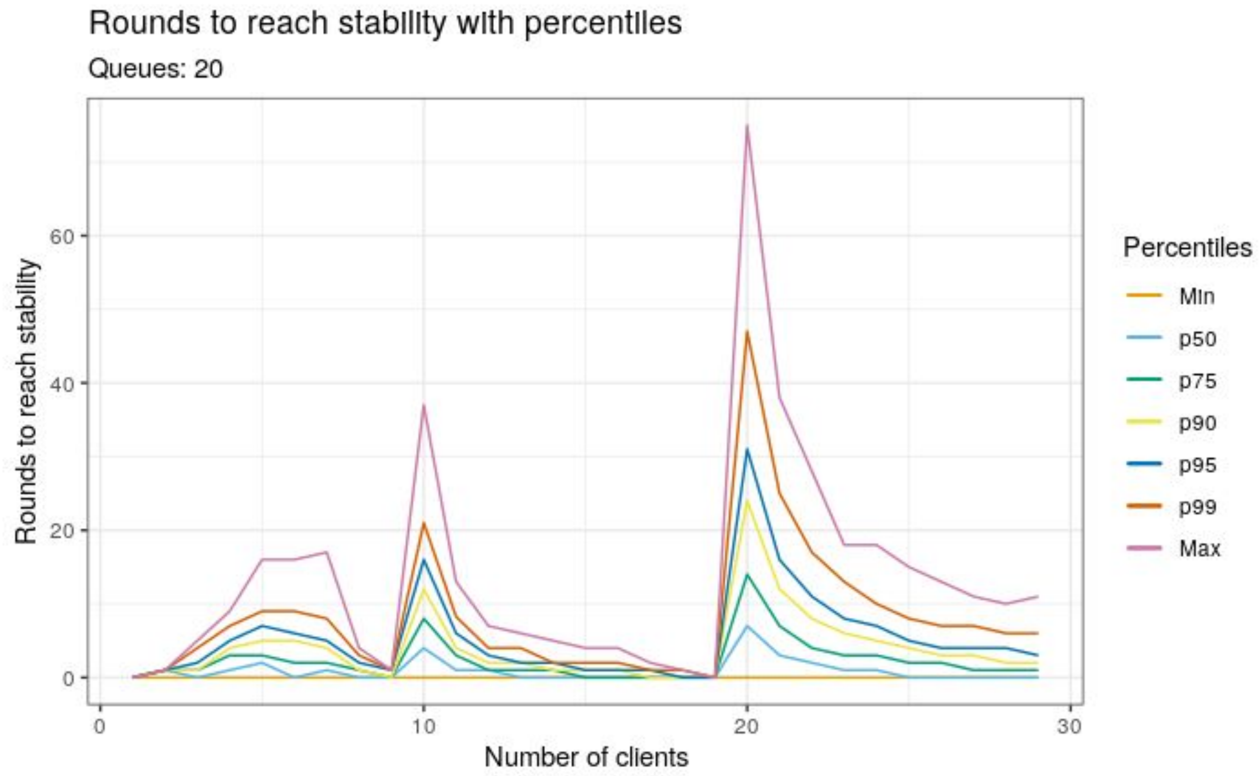
Dimension: Number of clients

Measured: Rounds to reach balance and stability

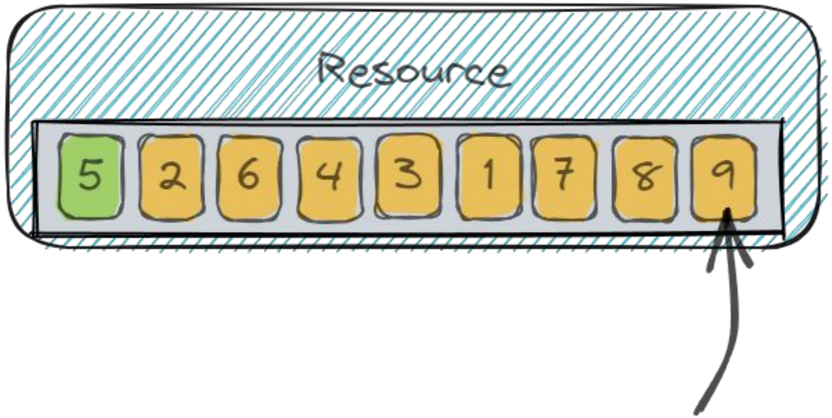
Each round is 30 seconds
=>
p99 of 25 minutes!



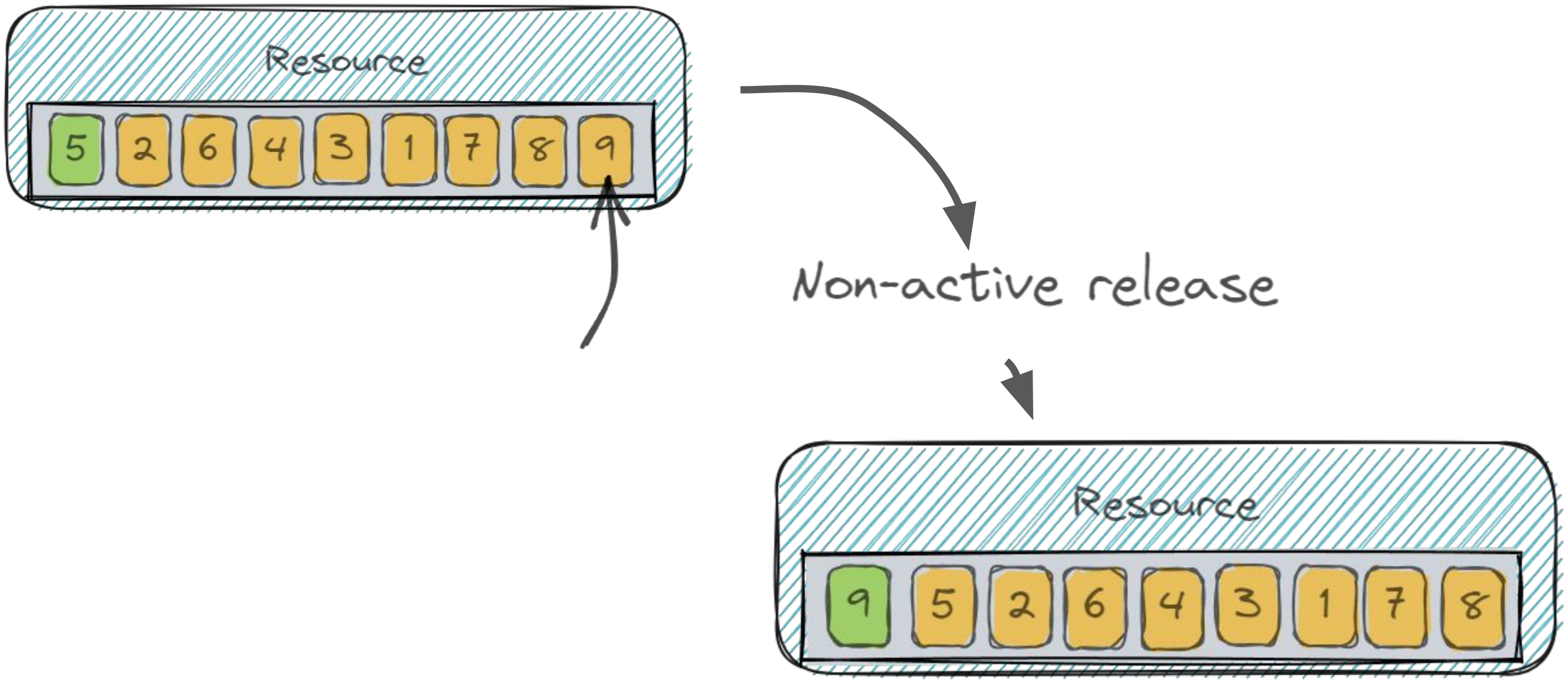
Scenario: One client dies
Dimension: Number of clients
Measured: Rounds to reach balance and stability



Why such variance?



Finding an optimization



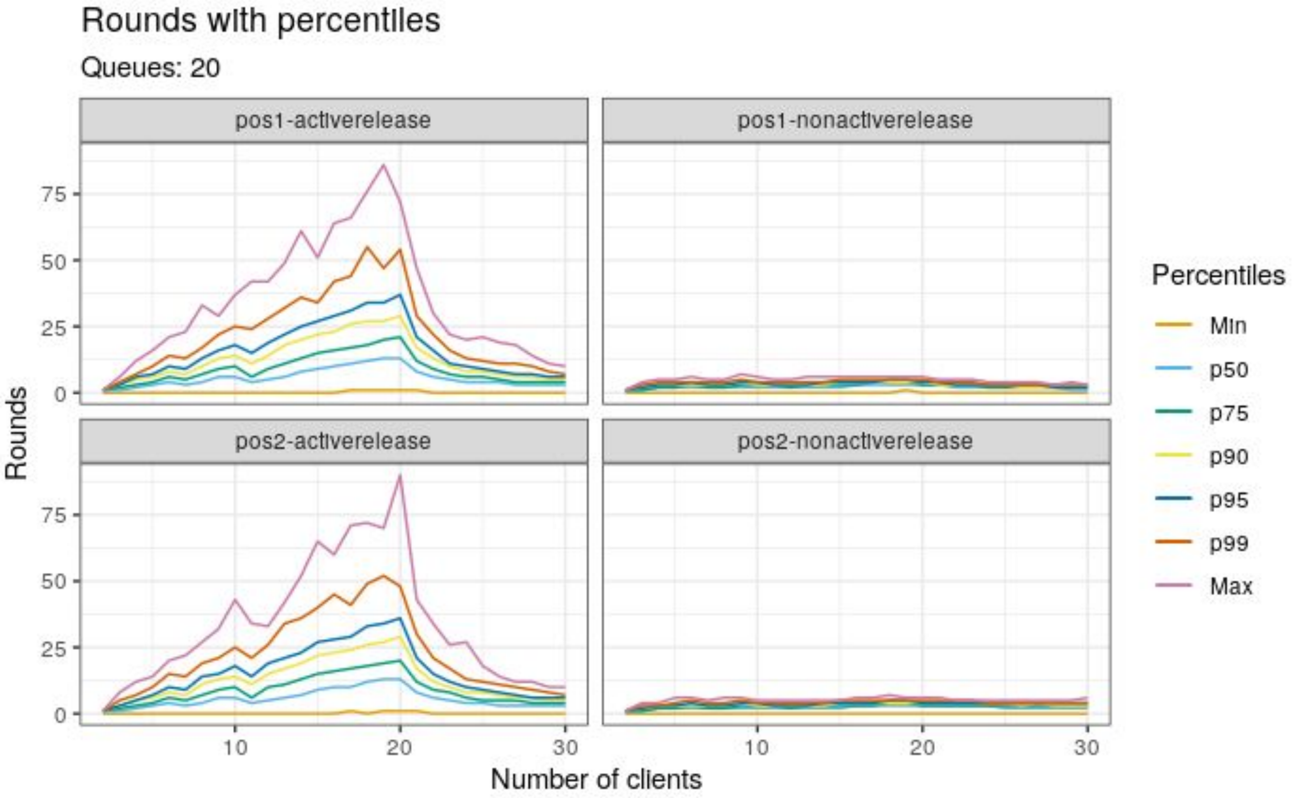
Scenario: Concurrent start-up of clients

Dimension: Number of clients

Measured: Rounds to reach balance and stability

Opt1:
Non-active
release

Opt2: Ranking
algorithm
(pos2)



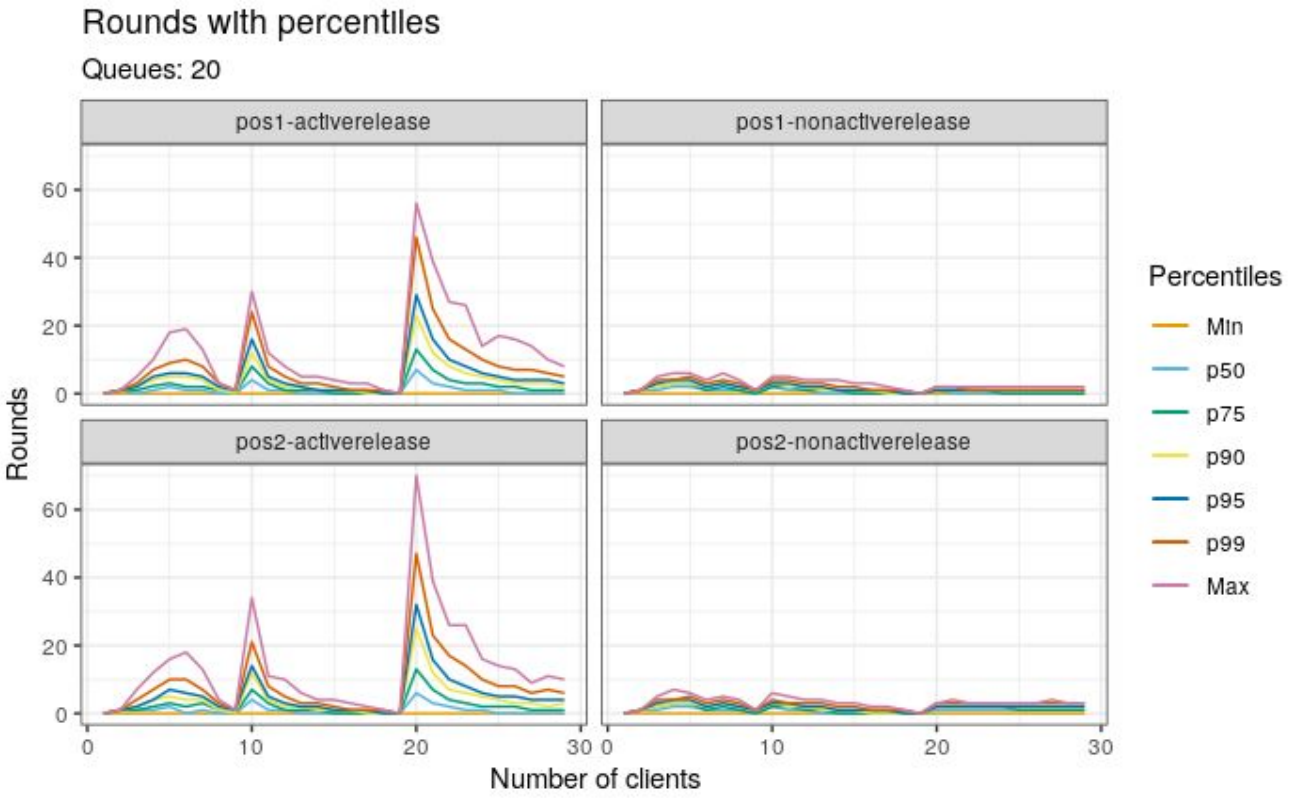
Scenario: One client dies

Dimension: Number of clients

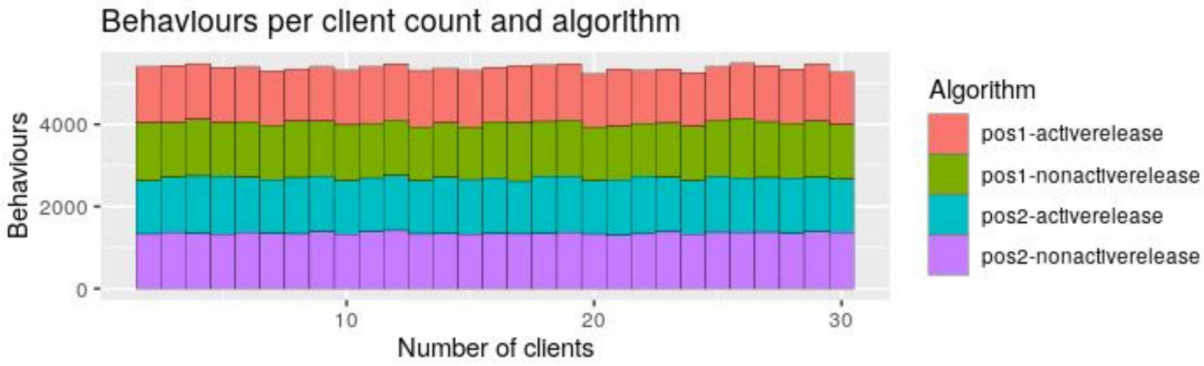
Measured: Rounds to reach balance and stability

Opt1: Non-active release

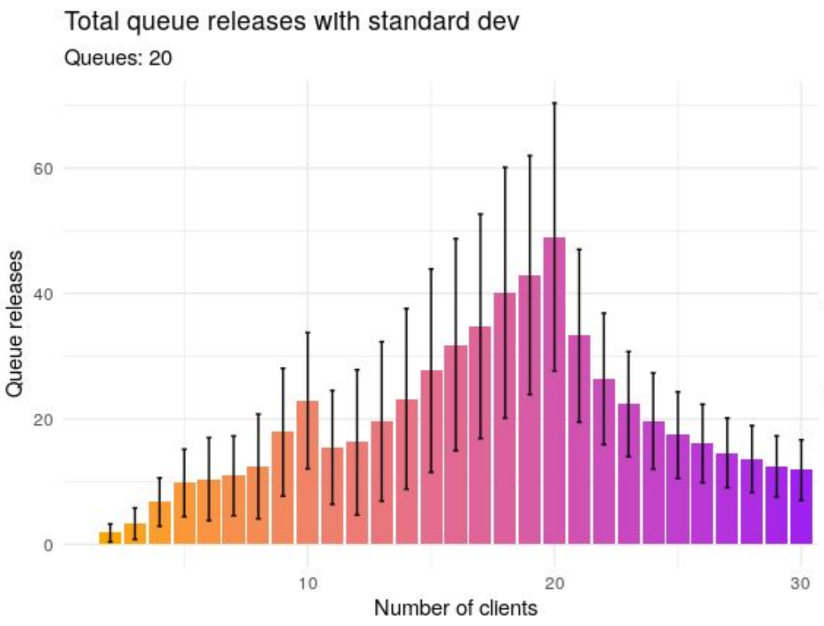
Opt2: Ranking algorithm (pos2)



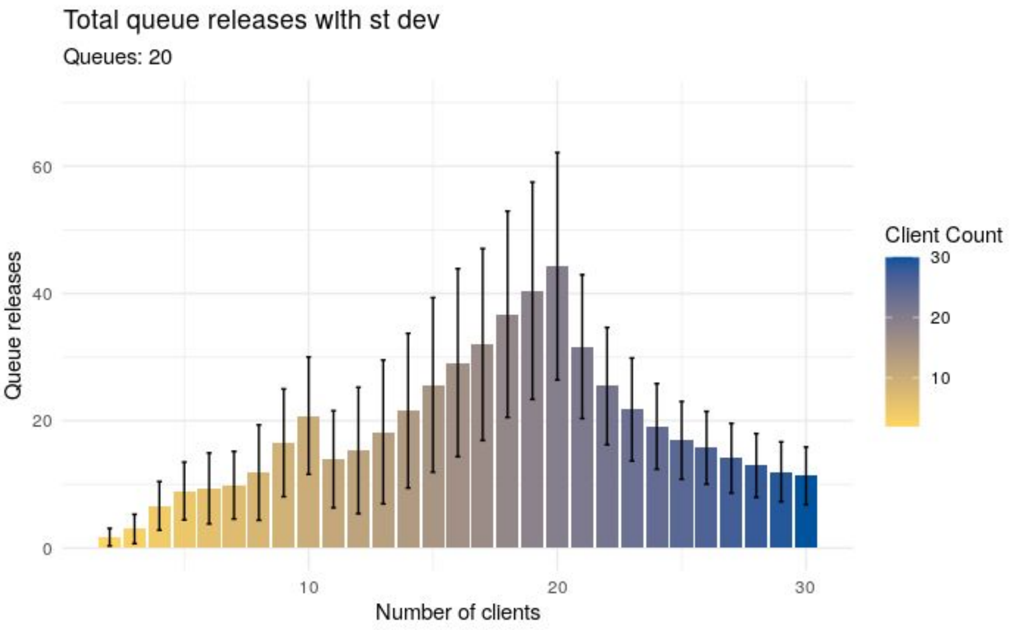
Checking dimension distributions



Comparing TLA+ data to original Python simulation

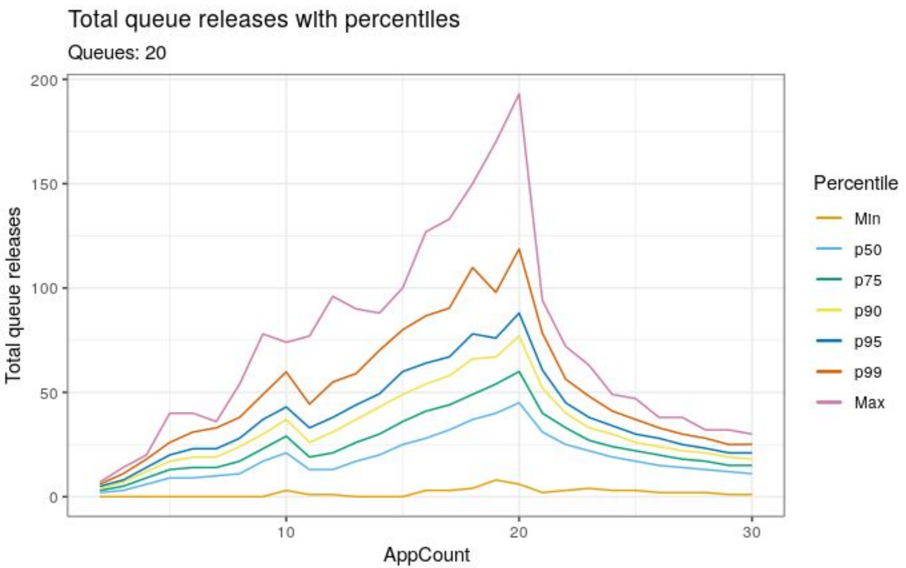


TLA+

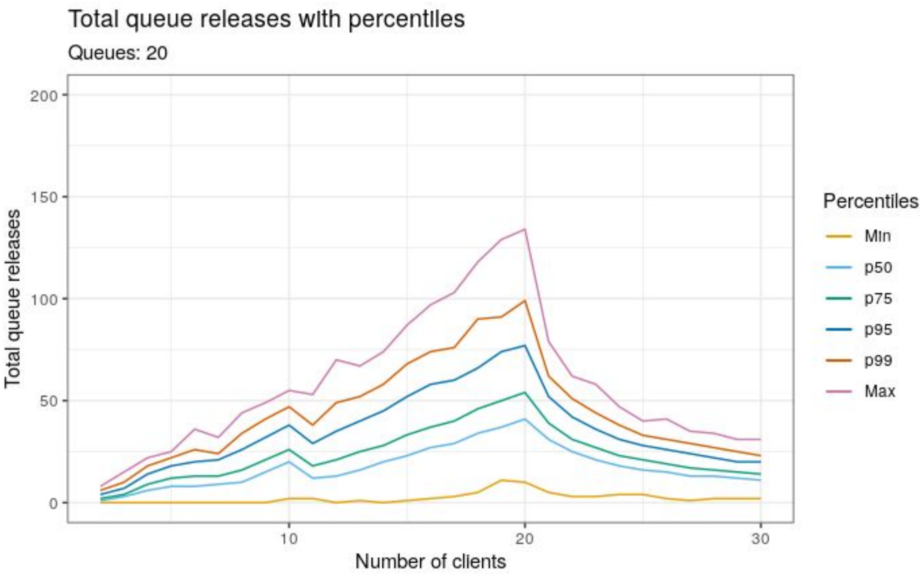


Python

Comparing TLA+ data to original Python simulation



TLA+



Python

Case Studies

(SWIM, RabbitMQ, Kafka)

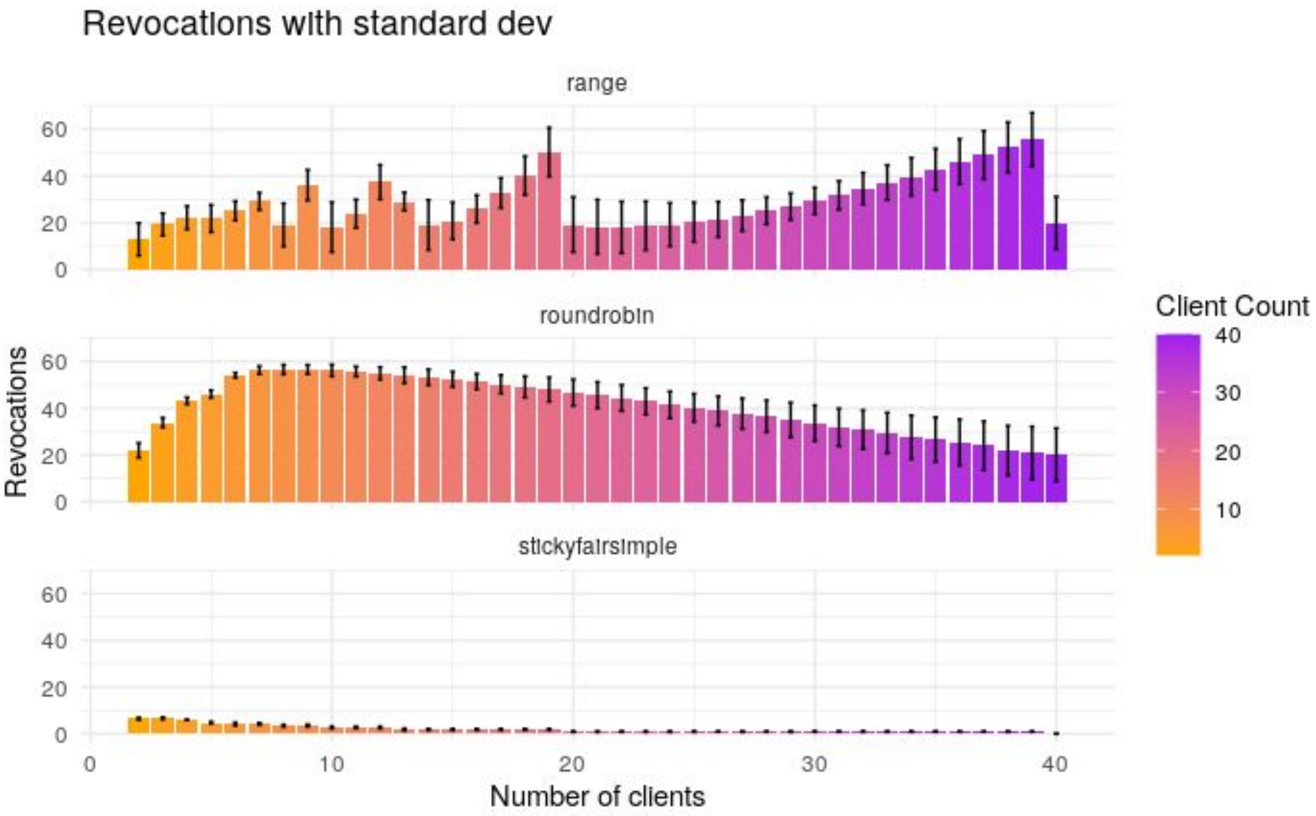
Kafka Group Rebalancing Protocol v3

- Leader based resource allocation
  - Multiple assignment strategies
- Low degree of non-determinism
- Design
  - Broker performs partition assignments by piggybacking on heartbeat messages
  - Strategies
    - Round-robin
    - Range
    - Sticky
  - Partition revocations disruptive
    - Minimize as much as possible

Kafka Group Rebalancing Protocol v3

Assignment strategies and revocations

20 partitions and killing a single client

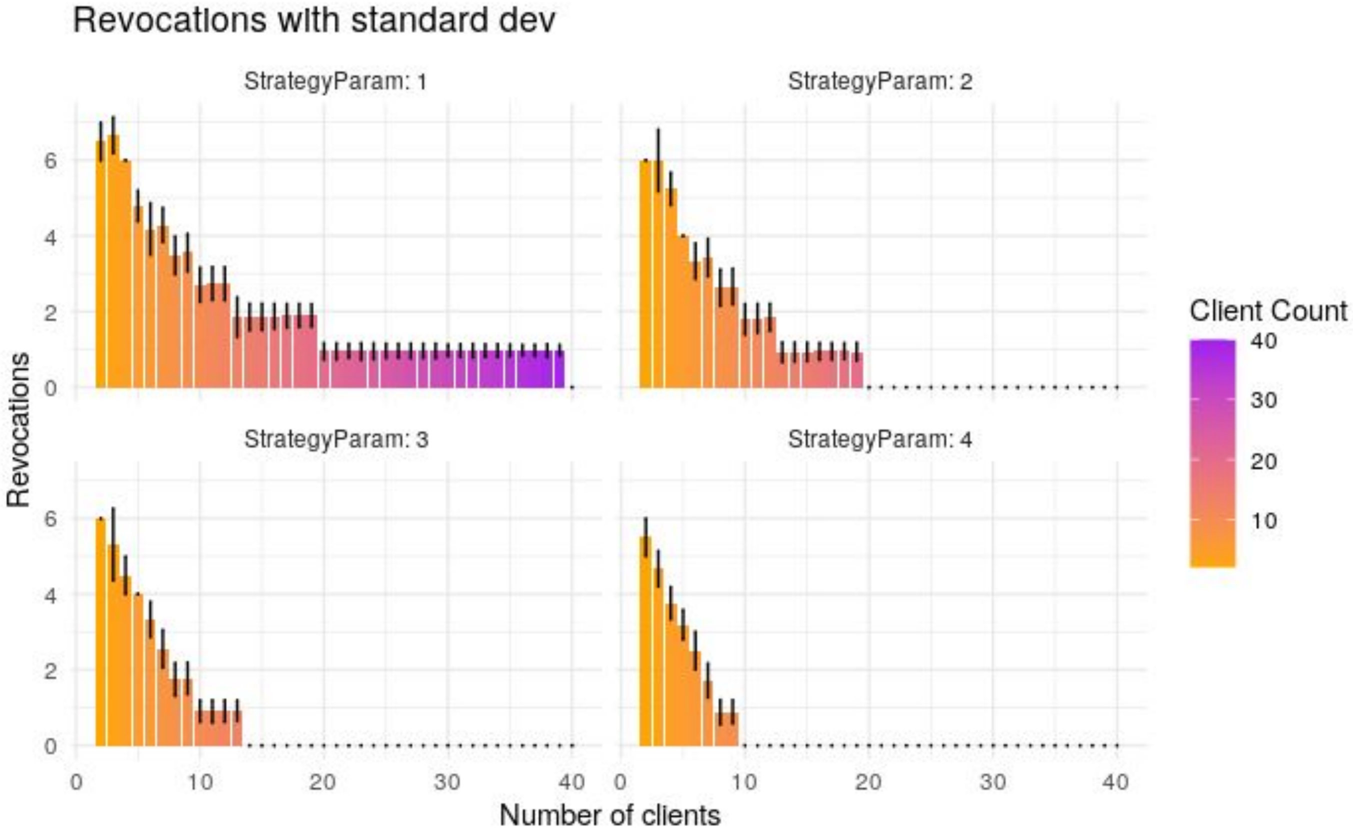


Kafka Group Rebalancing Protocol v3

Sticky assignment optimization: Distance to ideal assignment

Distances:
1-4

20 partitions and killing a single client



Conclusion

- Tried simulation successfully on 8 toy and real-world specs
  - Insights led to changes in RabbitMQ Reactor Streams client
- Unit of measure?
  - First-class citizen of spec :-)
  - System-level measures such as coherence & contention :-(
- *Define* the workload & perturbations of the system in TLA+
  - N% message loss, M dead nodes, W writes, …
  - Not via (non-machine-closed) fairness constraint :-(
- Scalability and Small scope hypothesis?
  - Larger number increase the resolution but do not seem to change the trend
  - Simulation is embarrassingly parallelizable!
  - TLC: Java Module Overrides (profiler), CommunityModules, TLCCache, Randomization.tla, …
    => keep talking at https://github.com/tlaplus/tlaplus/issues/601

Questions?

Q&A

Specs

- https://github.com/Vanlightly/formal-methods-playground/tree/master/tla/tlaplus-conf/swim
- https://github.com/Vanlightly/formal-methods-playground/tree/master/tla/tlaplus-conf/rabbitmq
- https://github.com/Vanlightly/formal-methods-playground/tree/master/tla/tlaplus-conf/kafka
- https://github.com/tlaplus/Examples/tree/master/specifications/KnuthYao
- https://github.com/tlaplus/Examples/tree/master/specifications/ewd998
- https://github.com/lemmy/ewd840/tree/mku-simulate-new
- System-level:
- https://github.com/lemmy/BlockingQueue/
- https://github.com/lemmy/PageQueue/

TLC Design Guidelines

- One language to rule them all
  - Define what is measured in TLA+
- Analysis orthogonal to TLA+
  - Integration with R, matplotlib, … via CSV/Json
  - But move more and more stats into TLA+
- “Wer misst misst Mist” (Who measures measures rubbish)
  - Environment and behavior validation in TLA+

TLC Changes

- TLC
  - Replace non-determinism with uniform probabilities in TLC in “-generate” mode
  - Built-in statistics in simulation mode
  - PostCondition
- TLC.tla
  - TLCGet(“config”)
  - TLCGet(“stats”)
  - TLCEval
  - TLCDefer (obsolete with -generate)
- TLCExt.tla
  - TLCTrace
- IOUtils.tla
  - IOEnv
  - IO[Env]Exec
- CommunityModules
  - CSV.tla
  - FiniteSetsExt.tla
  - Combinatorics.tla

TLCExt!TLCCache

- Introduce TLCCache operator
  - Its TLA+ definition / What is its parameter
  - Example where it is useful
- Contrast its performance benefits with a dedicated module override
  - https://github.com/Vanlightly/formal-methods-playground/issues/2
- Annotation-based module overrides