Lifeguard: Local Health Awareness for More Accurate Failure Detection

Armon Dadgar James Phillips Jon Currey
{armon,james,jc}@hashicorp.com

***Abstract*—SWIM is a peer-to-peer group membership protocol with attractive scaling and robustness properties. However, slow message processing can cause SWIM to mark healthy members as failed (so called false positive failure detection), despite inclusion of a mechanism to avoid this.**

We identify the properties of SWIM that lead to the problem, and propose Lifeguard, a set of extensions to SWIM which consider that the local failure detector module may be at fault, via the concept of *local health*. We evaluate this approach in a precisely controlled environment and validate it in a real-world scenario, showing that it drastically reduces the rate of false positives. The false positive rate and detection time for true failures can be reduced simultaneously, compared to the baseline levels of SWIM.

I. Introduction

Three key issues that any distributed system must address are discovery, fault detection, and load balancing among its components. Group membership is an intuitive abstraction that can be used to address all three issues simultaneously. Members of a group and its clients are offered a dynamically updating view of the current group membership, and use this view to perform actions such as request routing and state migration.

SWIM [1] is a group membership protocol with a number of attractive properties. It's peer-to-peer design and use of randomized communication make it highly scalable, robust to both node and network failures, and easy to deploy and manage. Its simplicity make it easy to implement and debug, compared to many distributed systems protocols.

We are aware of three mature open source implementations of SWIM. Butterfly [2] is part of Habitat [3], a popular software automation platform. Ringpop [4] was built to support the applications of a global transportation technology company. memberlist[5] is our implementation of SWIM, which underpins Consul[6], a popular service discovery and management tool, and Nomad[7], a high-availability, data center scale scheduler. Through our relationship with customers, we know of hundreds of thousands of running instances of Consul, and deployments with more than 6,000 members in a single group.

The SWIM paper identifies sensitivity to slow message processing as an issue with the basic SWIM protocol. Slow message processing can be caused by a wide variety of factors, including CPU contention, network delay or loss, and can lead SWIM to declare healthy members as faulty - so called *false positive failure detection*. To counter this, the SWIM paper proposes a Suspicion subprotocol, that trades increased failure detection latency for fewer false positives.

However, our experience supporting Consul and Nomad shows that, even with the Suspicion subprotocol, slow message processing can still lead healthy members being marked as failed in certain circumstances. When the slow processing occurs intermittently, a healthy member can oscillate repeatedly between being marked as failed and healthy. This 'flapping' can be very costly if it induces repeated failover operations, such as provisioning members or re-balancing data.

Debugging these scenarios led us to insights regarding both a deficiency in SWIM's handling of slow message processing, and a way to address that deficiency. The approach used is to make each instance of SWIM's failure detector consider its own health, which we refer to as *local health*. We implement this via a set of extensions to SWIM, which we call Lifeguard. Lifeguard is able to significatly reduce the false positive rate, in both controlled and real-world scenarios.

The rest of the paper is structured as follows: Section II motivates the advantages of SWIM that lead us to use it, and the kinds of scenarios where we have encountered this problem. Section III describes SWIM and memberlist, the implementation of SWIM that we use to evaluate Lifeguard. Section IV describes the Lifeguard extensions to SWIM. Section V describes the experimental evaluation of the components of Lifeguard, individually and in combination. Section VI describes Lifeguard's relationship to prior work. In Section VII we discuss the conclusions that can be drawn, and potential future work.

II. Motivation

SWIM uses randomized probe-based failure detection and gossip-based update dissemination to obtain a number of attractive properties:

- **Scalability.** In SWIM, the expected time to first detection of a failure, the false positive rate, and the message load per group member are independent of group size. Time to fully disseminate a failure grows logarithmically with group size.
- **Robustness.** Because the protocol is fully decentralized, the simultaneous failure or network partition of any subset of the group members can be tolerated. Even fully partitioned sub-groups can continue to operate, and will automatically merge once connectivity is re-established.

- **Ease of deployment and maintenance.** The fully decentralized nature of the protocol means a prospective member can contact any current member to join the group, and no special action has to be taken to keep the system healthy when a member leaves.
- **Simplicity of implementation.** The SWIM protocol has few states and messages. Because it is peer-to-peer, no special structure, such as leaders or hierarchies, has to be configured initially or maintained upon membership change.

The use of gossip-based update dissemination makes SWIM weakly consistent. That is, different members may have a different view of the group membership at a given point in time. In practice, weak consistency is acceptable for many applications. Where it is not acceptable, as the SWIM paper points out, strong consistency can be achieved by layering a consistent view on top of SWIM, that checkpoints the membership list.[1]

SWIM's failure detector sub-protocol is known to be susceptible to slow processing of its messages, which can result in false positive failure detections, where healthy members are incorrectly declared faulty. This is a serious concern, as there are often costs associated with diverting traffic away from a member, and with re-integrating it into the system once it is declared healthy again.

The SWIM paper addresses this issue by adding a Suspicion mechanism, which is explained in detail in Section III. However, our experience developing and supporting a range of systems that use SWIM has shown that even with the Suspicion mechanism, false failure detections can occur at a problematic rate under conditions sometimes experienced in data centers. The issue is exacerbated when multiple members are slow concurrently. Even healthy members may mark other healthy members as failed, if they are influenced by their interactions with the slow members.

Scenarios where we have debugged this issue include:

- Web servers that were provisioned for the steady state, but experience bursts of much heavier traffic.
- Ingress nodes that run firewalls and other edge services experiencing a sustained Distributed Denial of Service (DDoS) attack, leading to both high network & CPU load.
- Video transcode servers being assigned workloads that excessively oversubscribe the available CPU.
- Burstable Performance Instances, such as AWS T2 class and Azure B-Series virtual machines, being assigned workloads that exhaust their CPU credits, so that they are throttled by the hypervisor on which they are executing.

In these and other scenarios, the slow processing of SWIM messages on the affected machines led to healthy machines in the same membership group falsely being accused of failing. We have encountered this on bare metal and virtualized systems, in private data centers and public cloud environments.

Figure 1 shows the results of an experiment where we reproduce the characteristics of the video transcode scenario. We deploy a cluster of 100 single-core (`Standard_A1_v2` class) virtual machines into a region of the Microsoft Azure public cloud. The Consul agent (the daemon that must run on each node that is to be a member of the SWIM group) is deployed on all machines. We then run an extreme CPU-intensive workload on a subset of the machines. In this case we used the Linux `stress` tool, configured to run 128 processes, each of which executes a tight loop of math operations. The workload is run for 5 minutes. We log all member failure events raised by Consul during each test, and analyze the logs after the experiments are over to determine how many false positive failure detections occur.

The x-axis of Figure 1 shows the number of machines that have the `stress` workload running on them, ranging from 1 to 32 machines (where each machine represents 1% of the cluster). For each number of stressed machines, the y-axis shows two related metrics:

- **Total False Positives**. Failure events about healthy members (that did not have the `stress` workload running on them), that occur at any member, including the members running the stress workload.
- **False Positives at Healthy Members**. Failure events that are not only about healthy nodes, but were reported by healthy nodes. These are particularly concerning, as both of the agents involved - the one raising the event and the one that the event is about - are in fact healthy.

Each metric is shown twice. Once for Consul running unmodified SWIM and again for it running SWIM with Lifeguard.

Figure 1 shows that for SWIM, even a single overloaded member is sufficient to cause some false positive failure detection events, and as few as 4 overloaded members (representing 4% of the cluster) is enough to produce hundreds of false positives at healthy members, with the problem becoming more severe as the number of stressed members increases. By contrast, Lifeguard does not produce false positives until 16 machines are stressed concurrently, and false positives at healthy members until 32 machines are stressed concurrently. Both are produced at significantly lower levels than with SWIM.

These problems do not occur frequently. But when they do, they can be highly disruptive. Investigating a number of such incidents that had been escalated as high-priority support requests led to the development of Lifeguard.

III. SWIM AND MEMBERLIST

In this section we first review SWIM, and then describe memberlist, the implementation of SWIM with which we evaluate Lifeguard .

A. SWIM

SWIM has two components:

- a **Failure Detector**, that detects failures of members.

[1]This is the approach[8] taken by Consul[6], which uses Raft[9] to present a strongly consistent view of the group membership it obtains from memberlist[5]. While this produces a dependency on a quorum of the Raft servers, the benefits of SWIM described above still apply to the resources under management, which only need to act as SWIM group members.

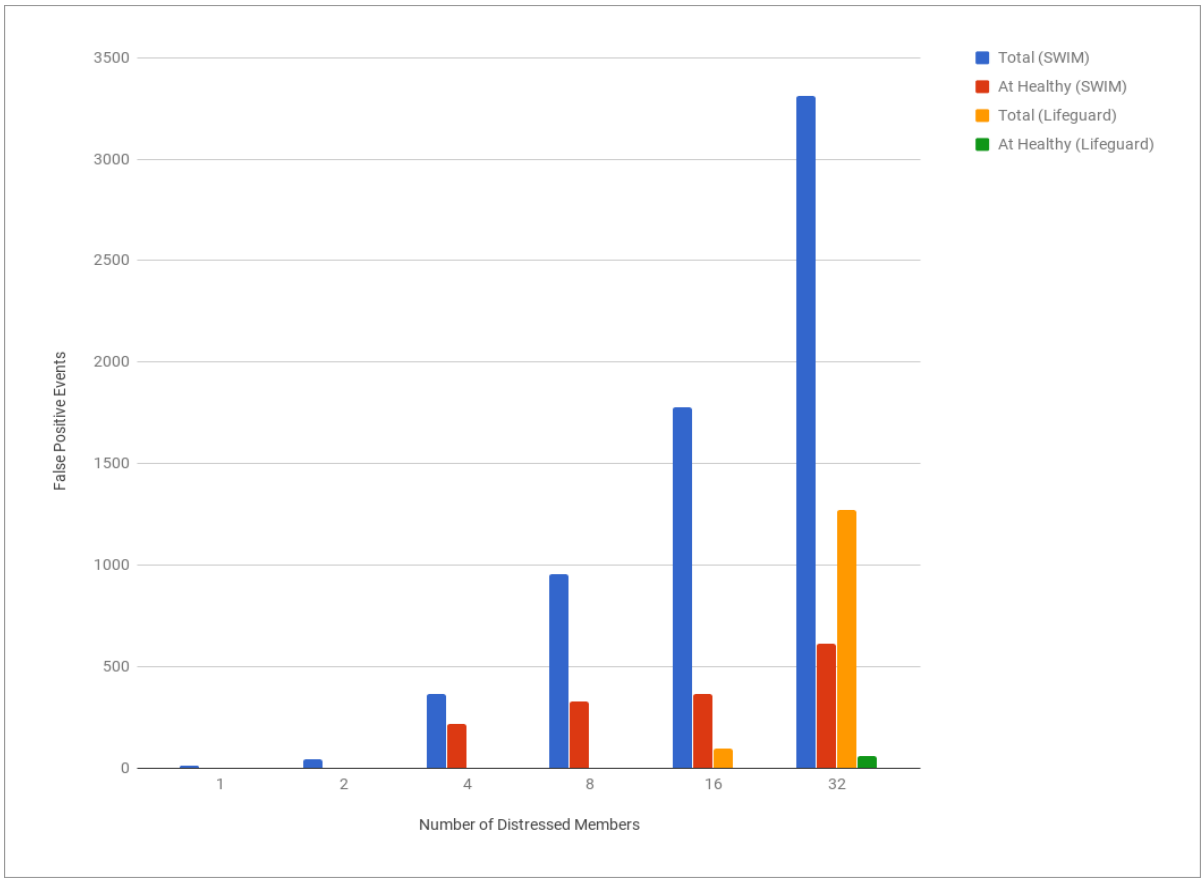

Fig. 1. **False Positives from CPU exhaustion.** The total number of false positive failure detection events and the number occurring at healthy members, as the number of distressed members is varied from 1 to 32. Results are shown for unmodified SWIM and SWIM with Lifeguard.

- a **Dissemination Component**, that disseminates updates about members that joined or left the group, or failed.

The failure detector is taken from prior work[10]. It is fully decentralized, with each group member working asynchronously in rounds of some configurable duration, called the *protocol period*. In each protocol period, each member picks one other member at random to check the health of, and performs a *direct probe* by sending that member a `ping` message. If an `ack` message is not received within a configurable amount of time, the member initiating the check performs an *indirect probe*, by choosing k more members and sending each of them a `ping-req` message. Receiving the `ping-req` message causes each of the k members to send a `ping` message to the member under investigation. If any of them receives an `ack` in response, it forwards it to the original probing member. If the original member does not receive any `ack` messages from the direct or indirect probe by the end of the protocol period, the probed member is considered to have failed the failure detection.

The dissemination component is gossip-based.[2] Each update about a member joining or leaving the group or failing is shared with one other member $\lambda log(n)$ times, where n is the size of the known group and λ is a tunable multiplier. The updates are piggybacked on the `ping`, `ping-req` and `ack` messages of the failure detection protocol, so that no additional messages are sent. The number of updates piggybacked on each message is limited (to respect any limit on the message size, such as the MTU of a UDP packet), and updates that have been shared less times are preferred, to try and progress the dissemination of all updates in times of high update activity.

In the simplest realization of SWIM, when a member fails a failure detection check, it is immediately marked as failed, and the failure is shared with the group via the dissemination component. However, the SWIM authors themselves observe false positive failure detections, and cite slow processing messages as the primary cause. To address this, the SWIM paper introduces the *Suspicion* mechanism. It is introduced as

[2]The SWIM paper tentatively proposes a multicast based dissemination component, but immediately rejects that approach in favor of the gossip-based approach.

an extension, but in practice is a necessary part of SWIM. It is implemented by all three of the mature SWIM implementations discussed in Section I.

With Suspicion enabled, a member that fails a failure detector check goes to an intermediate *suspected* state, and a `suspect` message is gossiped via the dissemination mechanism, to see if the suspicion can be refuted before a suspicion timeout is reached. Any member that receives a `suspect` message also marks the specified member as suspect, and gossips its suspicion. If a suspicion timeout is reached without the suspicion being refuted, the suspected member is declared faulty, by the gossiping of a `confirm` message. In this way, the Suspicion mechanism trades increased failure detection latency for a lower false positive failure detection rate.

A suspicion is refuted by an `alive` message being gossiped about the suspected member and reaching all members that harbor the suspicion before any of them reaches its suspicion timeout. The SWIM paper describes two mechanisms by which an `alive` message may be originated: Either by a member that harbors a suspicion, after it successfully probes the suspected member in a round of failure detection, or by the suspected member itself, after it receives a `suspect` or `confirm` message about itself. However, in practice, the suspected member must gossip an `alive` message about itself for refutation to work.[3]

The other refinement to the basic protocol that the SWIM paper makes is to have each member select its fault detector targets in round-robin fashion from its list of known members, as opposed to completely at random. Without this, the worst-case first-detection latency would be unbounded, due to the (extremely rare) case that selection of fault detector targets across all members of the group repeatedly fails to select the faulty member. By probing in round-robin, the worst case is bounded. However, each member's list still has a random order, with new members being inserted at random positions. Consequently, the *expected* first detection latency is unchanged.

B. memberlist

memberlist[5] is an open source implementation of SWIM, used by tools including Consul[6], Nomad[7], and Serf[11]. memberlist implements all of the features of SWIM described above. It has the following additional features:

- memberlist's fault detector uses UDP by default for both direct and indirect probes. But in parallel to issuing indirect probes over UDP, it will attempt a direct probe over TCP. This helps with situations where TCP traffic is correctly routed, but UDP is not, which is a pathology sometimes encountered in network configuration.
- memberlist adds an *anti-entropy* mechanism, by which each member periodically does a full state sync with another randomly selected member, over TCP. The *push-pull* approach from [12] is taken, with incarnation numbers used to reconcile conflicting state about a given member. This full state sync increases the likelihood that nodes are fully converged more quickly, at the expense of more bandwidth usage. It is particularly helpful for speeding up recovery from a network partition.
- memberlist has a dedicated gossip layer separate from the failure detection protocol. Like SWIM, memberlist will piggyback gossip messages (`suspect`, `alive` and `confirm`[4]) on to fault detector messages (`ping`, `ping-req`, `ack`), but it also will periodically send out gossip messages on their own. This allows the gossip rate to be tuned independently of the failure detection rate, and if necessary faster than it, to speed the rate of convergence.
- memberlist retains the state of failed nodes for a period of time, so that information about failed nodes can be passed in a full state sync. This helps the state of the group converge more quickly.

As these features are typically enabled in deployments of memberlist, they are all enabled for the evaluation in this paper. Butterfly [2] and Ringpop [4] implement many of the same additional features.

IV. Lifeguard

While investigating possible solutions to the problems described in Section II, we observed that the Suspicion mechanism still assumes some timely processing of messages. In particular, refutation of a suspicion can only succeed if the refuting `alive` message is processed in a timely manner by all members suspecting that member. Therefore slow processing by the failure detector module itself is the primary cause of the false positives that SWIM's Suspicion mechanism fails to suppress.

We also observed that missing expected responses could indicate a member is experiencing slow message processing, and that an episode of slow message processing at a given group member is likely to impact multiple of its interactions with other members in a short period of time. We think of these as measures of the health of the local failure detector instance at that member, which we call *local health* for short.

Based on these insights, we designed Lifeguard: a set of extensions to SWIM which make it into an adaptive protocol. Lifeguard uses heuristic measures of *local health* to let a member consider when its failure detector might be slow processing messages, and if so to dynamically adjust its timeouts to mitigate timeliness issues.

Lifeguard consciously retains the same design as SWIM so far as possible. It differs from SWIM only in three components, that provide its novel behavior:

[3] `suspect`, `confirm` and `alive` messages carry an incarnation number for the member they are about, to establish a precedence for competing messages, and guide the state of the group towards convergence. As section 4.2 of the SWIM paper points out, `alive` messages only override the other message types if they have a higher incarnation number, and only the suspected member can increment its incarnation number. It does so in response to receiving a gossiped `suspect` or `confirm` message about itself.

[4] In memberlist, the `confirm` message is renamed to `dead`, as the name `confirm` is ambiguous as to what is being confirmed.

- **Local Health Aware Probe (LHA-Probe)** replaces the probing stage of SWIM's failure detector, which has a fixed probe period and timeout, with one where they are dynamically adjusted, based on that member's recent failure detection-related communication with other members.
- **Local Health Aware Suspicion (LHA-Suspicion)** replaces the suspicion stage of SWIM's failure detector, which has fixed suspicion timeouts, with one that has dynamic timeouts. The timeout for each new suspicion starts significantly higher than it would in the fixed case, but is reduced as independent suspicions about the same suspected member are processed.
- **Buddy System** replaces SWIM's piggyback message selector with one that prioritizes notifying a suspected member of the suspicion, to reduce the average time to refutation, which helps both the Local Health Aware Probe and Local Health Aware Suspicion components be even more effective.

These components are described in detail the sections that follow.

A. Local Health Aware Probe

Local Health Aware Probe (LHA-Probe) replaces the probing stage of SWIM's failure detector, which has a fixed probe period and timeout, with one where they are dynamically adjusted, based on that member's recent failure detection-related communication with other members. Several sources of feedback are used:

- The number of `ack` messages that have been received is compared to the number of `ping` and `ping-req` messages issued. Missing `ack` messages could be due to local slowness, especially if there are multiple.
- The need to refute a suspicion against itself indicates that the member may not have processed recent `ping` messages in a timely manner.
- Local Health Aware Probe adds a `nack` message to the fault detector protocol, which is sent in the case of failed indirect probes.[5] This gives the member that initiates the indirect probe a way to check if it is receiving timely responses from the *k* members it enlists, even if the target of their indirect pings is not responsive.

These different sources of feedback are combined in a Local Health Multiplier (LHM). LHM is a saturating counter, with a max value S and min value zero, meaning it will not increase above S or decrease below zero. The following events cause the specified changes to the LHM counter:

- Successful probe (`ping` or `ping-req` with `ack`): -1
- Failed probe +1
- Refuting a suspect message about self: +1
- Probe with missed `nack`: +1

[5]When a member is sent a `ping-req` message, it will send a `nack` back at 80% of the probe timeout unless it receives an `ack` by that time. An `ack` is still forwarded if it is received after the `nack` has been sent, and a member receiving a `nack` followed by an `ack` within the timeout period considers this as a successful indirect probe.

The current value of LHM is used to set the probe interval and timeout as follows:

$$ProbeInterval = BaseProbeInterval.(LHM(S) + 1)$$
$$ProbeTimeout = BaseProbeTimeout.(LHM(S) + 1)$$

ProbeInterval is the period between attempting a liveness probe against successive randomly selected peers, and *ProbeTimeout* is the timeout on receiving an `ack` to a given probe. In the memberlist implementation, we set *BaseProbeInterval* to 1 second and *BaseProbeTimeout* to 500 milliseconds. *S* defaults to 8, which means the probe interval and timeout will back off as high as 9 seconds and 4.5 seconds, respectively.

B. Local Health Aware Suspicion

Local Health Aware Suspicion (LHA-Suspicion) replaces the suspicion stage of SWIM's failure detector, which has fixed suspicion timeouts, with one that has dynamic timeouts. The timeout for each new suspicion starts significantly higher than it would in the fixed case, but is reduced as independent suspicions - from other members, but about the same suspected member - are processed. In this way, the timeout will fall to its minimum level as long as the local member is receiving and processing gossip messages in a timely manner. Conversely, the suspicion timeout will remain high for members that are not receiving and processing gossip messages in a timely manner.

The timeout for a given suspicion is calculated as follows:

$$SuspicionTimeout = max\left(Min, Max - (Max - Min)\frac{log(C+1)}{log(K+1)}\right)$$

where:

- Min and Max are the minimum and maximum Suspicion timeout. See Section V-C for discussion of their configuration.
- K is the number of independent suspicions required to be received before setting the suspicion timeout to Min. We default K to 3.
- C is the number of independent suspicions about that member received since the local suspicion was raised.

The timeout is recalculated whenever a `suspect` message is received that represents a previously unseen independent suspicion about the same member. At that time, the current suspicion timer is canceled and replaced with one for the remaining time until the new reduced timeout. If that amount of time has already passed, the timeout is triggered.

Logarithmic decay is used so that each successive reduction in the timeout is smaller than the last, as more independent `suspect` messages are received. The intuition behind this is that the first independent message gives the biggest increase in confidence that messages are being received in a timely manner, with each subsequent message adding less to the confidence.

To make independent suspicions more prevalent, when LHA-Suspicion is enabled the first K independent suspicions

received about the same member are re-gossiped. Each suspicion is sent $\lambda log(n)$ times, so that if K more suspicions are received, the maximum number of messages sent is $(K + 1)\lambda log(n)$. Without LHA-Suspicion, the independent suspicions are not re-gossiped, and only the member's own suspicion is gossiped, a maximum of $\lambda log(n)$ times.

C. Buddy System

In SWIM, a suspected member is not guaranteed to hear of the suspicion at the first opportunity. A suspected node only learns of the suspicion when it receives a gossiped `suspect` message about itself. While gossip messages are piggybacked on fault detector messages, including `ping` messages, the rules governing the dissemination of gossip messages include a limited number of gossip messages per piggyback, limited re-sends of each gossip message, and a preference for newer gossip messages.

Buddy System replaces SWIM's piggyback message selector with one that prioritizes notifying a suspected member of the suspicion. This guarantees that any node that pings a suspected node (either on its own behalf, or for the indirect path of another node) will communicate the suspicion as part of the ping. This can result in refutation starting sooner, which would be helpful even without the other Lifeguard components. But it also helps Local Health Aware Probe and Local Health Aware Suspicion work more effectively.

V. Evaluation

A. Evaluation Criteria

We evaluate Lifeguard according to the same criteria used to evaluate SWIM in the original paper (see Section 5 of [1]). Namely:

- **Failure Detection False Positives.** Lifeguard sets out to reduce the number of healthy members that are mistakenly marked as failed.
- **Detection and Dissemination Latency.** Lifeguard should not increase the time to first detection or full dissemination of true positive fault detection.
- **Message Load.** We consider the number of messages and bytes sent. Lifeguard should either decrease these, or not increase them by very much.

B. Configurations Tested

The three components of Lifeguard described in Section **??** are evaluated separately and in combination, in order to understand their relative contribution, and the way they interact with one another.

Experiments are run for each configuration described in Table I. The '`SWIM`' configuration gives the performance with Lifeguard completely disabled. It is the baseline against which the other combinations are compared. This approach is made possible by running a modified version of Consul, where each component can be enabled or disabled independently.

C. Suspicion Timeout Configuration

As described in Section IV-B, Lifeguard's Local Health Aware Suspicion component makes use of a Min and Max Suspicion timeout. In the memberlist implementation, these are configured as follows:

$$Min = \alpha log_{10}(n) ProbeInterval$$
$$Max = \beta Min$$

where:

- *n* is the number of members in the known group.
- *ProbeInterval* is the interval between successive failure detector probe messages. The default value of 1 second is used for all experiments.
- α and β are tunable parameters.

To examine the effect of the tunable parameters, each experiment is repeated nine times, with full Lifeguard (test configuration `Lifeguard`) configured with a different combination of $\alpha = 2, 4, 5$ and $\beta = 2, 4, 6$. The performance of the different combinations is compared with the baseline performance of memberlist with Lifeguard completely disabled (test configuration `SWIM`), which has a fixed Suspicion timeout, equivalent to configuring $\alpha = 5$ and $\beta = 1$.

D. Experiments

The SWIM paper correctly identifies that slow message processing may be due to a number of factors, including CPU exhaustion, network delay, and packet loss - either at the local host or in the network. The net result is always failure to process one or more protocol messages in a timely manner. For the purpose of this investigation, we induce slow message processing by pausing the sending and receiving of protocol messages at selected group members for well defined periods of time. We call each period of delay at one member an anomaly.

Two different types of experiment are used to evaluate the criteria defined in section V-A: Threshold and Interval. They are described in the subsections that follow. The reason for two types of experiments is as follows:

- The Threshold experiment introduces a single set of concurrent anomalies per experiment. This allows the latency from the start of an anomaly to its detection and dissemination to be examined, as with only a single set of anomalies, the causality is clear.
- However, in real-world situations, CPU and network delays can be intermittent, with processes making progress in small bursts. The Interval experiment explores this space by introducing anomalies in a cyclic way for the duration of each experiment. The duration of and interval between anomalies is varied across a number of different experiments.

The experiments are performed using deployments of Consul, a service discovery and monitoring system built on top of memberlist. However, none of the higher-level features of Consul (such as a Raft[9]-based consistent view of the available services) are employed, and the cluster is deployed without

Configuration	Description
SWIM	Regular SWIM
LHA-Probe	SWIM + Local Health Aware Probe
LHA-Suspicion	SWIM + Local Health Aware Suspicion
Buddy System	SWIM + Buddy System
Lifeguard	All Lifeguard components enabled

TABLE I
CONFIGURATIONS TESTED.

server instances, so that only the features of memberlist are exercised.

1) Threshold Experiment: The Threshold experiment is used to examine the effect of Lifeguard on detection and dissemination latency. It has the following form:

- 128 Consul agents are started in a single Linux VM, communicating over the loopback network interface.
- 15 seconds are allowed for the agents to quiesce.
- C instances (selected at random) enter an anomalous state, where they block immediately before sending or after receiving any protocol message from another member of the cluster. [6]
- The anomalies continue for a duration D, at the end of which the blocked sends and receives are unblocked.
- The experiment continues until all 128 Consul instances return to seeing one another as healthy, or until 120 seconds have passed from the start of the experiment.

Many instances of the Threshold experiment are run for each configuration tested, sweeping a range of values for C and D. The values tested are given in Table II. The experiment is run 10 times for each combination of Lifeguard components and other experiment parameters.

2) Interval Experiment: The Interval experiment is used to examine the effect of Lifeguard on both false positive failure detection, and message load. It has the same form as the Threshold experiment, apart from the following differences:

- At the end of the anomalous period of duration D, each of the anomalous Consul instances returns to normal operation for an interval I.
- The cycle of anomalous and normal operation repeats in rotation, for periods of length D and I respectively, until at least 120 seconds have passed since the beginning of the test. The test ends at the end of the next anomalous period.

Many instances of the Interval experiment are run for each Lifeguard configuration tested, sweeping a range of values for C, D and I. The values tested are given in Table III. The experiment is run 10 times for each combination of Lifeguard components and other experiment parameters.

[6]The start and end of the anomaly period are synchronized via the system clock of the VM, so that the C anomalous instances change state in lock-step. While many more combinations of start and end time could be examined, this represents the worst case of C fully correlated anomalies, such as from power loss to a rack.

E. Experiment Environment

The experiments are run on Microsoft Azure Compute-Optimized (F-Series) VMs, which are deployed on 2.4 GHz Intel Xeon E5-2673 v3 (Haswell) processors. F16 instances are used, which are each allocated 16 cores and 32GiB of RAM. Ubuntu 16.04 LTS daily build 201701280 is used, and Consul is configured to write DEBUG-level logs to /dev/shm, the ramdisk that Ubuntu configures by default. Logs are copied to SSD at the end of each experiment.

To reduce scheduling and memory access indeterminacy, 8 Consul agents are pinned to each of the 16 CPU cores and the associated memory bank, using the Linux `numactl` tool. CPU usage is monitored throughout the lifetime of each experiment, by sampling `/proc/stat` [13] at a 1 second interval, and it is confirmed that there is spare CPU capacity in all experiments, indicated by a increase in the aggregate core idle time at each interval. In practice, 16 cores on this class of CPU is excessive for 128 Consul agents.

F. Results

The experiments explore a large combinatorial space of parameter values. To make the results more tractable, we first examine the performance of Lifeguard with the tunable Suspicion timeout parameters (described in Section V-C), set to the highest values considered: $\alpha = 5$ and $\beta = 6$. We then examine the effect of lowering α and β.

1) Failure Detection False Positives: The Interval experiment, described in Section V-D2, is used to measure the effect of Lifeguard on the occurrence of failure detection false positives - that is, of healthy agents mistakenly being marked as failed. We define a failure detection false positive as occurring each time an agent failure event is raised about a Consul agent that is not in the set of agents for which anomalies have been introduced. Within these false positives, we distinguish between false positives that occur at any Consul agent (denoted FP), and those that occur at healthy agents (denoted FP−). FP− are most concerning, as in this case, both of the agents involved - the one raising the event and the one that the event is about - are in fact healthy.

Table IV gives the aggregated false positive statistics for all Interval experiments where $\alpha = 5$ and $\beta = 6$. The meaning of each column is as follows:

- `Configuration Tested` : Combination of Lifeguard components enabled, as described in Section V-B.

Parameter	Label	Values Tested
Concurrent anomalies	C	1, 4, 8, 12, 16, 20, 24, 28, 32
Duration of each anomaly	D	128, 512, 2048, 8192, 16384, 32768

TABLE II
THRESHOLD EXPERIMENT PARAMETERS AND VALUES TESTED. DURATIONS ARE GIVEN IN MILLISECONDS.

Parameter	Label	Values Tested
Concurrent anomalies	C	1, 4, 8, 12, 16, 20, 24, 28, 32
Duration of each anomaly	D	128, 512, 2048, 8192, 16384, 32768
Interval between anomalies	I	1, 4, 16, 64, 256, 1024, 4096, 16384

TABLE III
INTERVAL EXPERIMENT PARAMETERS AND VALUES TESTED. DURATIONS AND INTERVALS ARE GIVEN IN MILLISECONDS.

- `FP Events` : Total number of false positive failure events occurring at all Consul agents.
- `FP- Events` : Number of false positive failure events occurring at healthy agents (outside of the set that have anomalies introduced).
- `FP % SWIM` : `FP Events` as a percentage of the value for `SWIM` (the baseline).
- `FP- % SWIM` : `FP- Events` as a percentage of the value for `SWIM` (the baseline).

Configuration Tested	FP Events	FP- Events	FP % SWIM	FP- % SWIM
SWIM	339002	1326	100.00	100.00
LHA-Probe	229574	436	67.72	32.88
LHA-Suspicion	10174	89	3.00	6.71
Buddy System	318935	591	94.08	44.57
Lifeguard	5193	25	1.53	1.89

TABLE IV
AGGREGATED FALSE POSITIVE RESULTS FOR ALL EXPERIMENTS WHERE $\alpha = 5$ AND $\beta = 6$. FOR EACH CONFIGURATION TESTED, `FP EVENTS` IS THE TOTAL NUMBER OF FALSE POSITIVE EVENTS, AND `FP- EVENTS` IS THE NUMBER OF FALSE POSITIVE EVENTS AT HEALTHY NODES. `FP % SWIM` AND `FP- % SWIM` GIVE THE SAME RESULTS AS THE PERCENTAGE OF THEIR RESPECTIVE VALUES FOR `SWIM`.

Table IV shows that false positives are dominated by those occurring at slow processing members. This is indicated by `FP-` being a small proportion of `FP`, for all configurations tested, including the baseline with Lifeguard completely disabled (`SWIM`).

Table IV also shows that the false positive rate is drastically reduced by the introduction of Lifeguard. All components of Lifeguard contribute to the reduction, with Local Health Aware Suspicion (`LHA-Suspicion`) making the biggest individual contribution. Combining all of the components (`Lifeguard`) has the greatest effect. Both the overall number of false positives (`FP Events`) and false positives at healthy nodes (`FP- Events`) are reduced to less than 2% of the baseline levels for SWIM. This represents a more than 50x reduction in false positives.

The effect of Buddy System (`Buddy System`) is noteworthy, since it more than halves the false positives at healthy members (`FP-`), but has relatively little effect on the overall number of false positives (`FP`). This difference is explained by considering its method of action - helping a suspected member become aware of the suspicion in a more timely manner. This in turn can lead to refutation starting sooner. Healthy members (responsible for `FP-`) can receive and process the refutation in a timely manner, where as members experiencing slow message processing often can not. Since `FP` is in general dominated by false positives members experiencing slow message processing, this leaves it little changed by Buddy System.

The results in Table IV aggregate the false positive event counts for all tested numbers of concurrent anomalies (`C`, as defined in Section V-D1). Figures 2 and 3 consider the variation in number of false positives with the number of concurrent anomalies.

Figure 2 shows the total number of false positives (`FP Events`) for each number of concurrent anomalies tested. It shows clearly that the number of false positives rises with the number of concurrent anomalies, but that at every concurrency level, full Lifeguard (`Lifeguard`) reduces the number of false positives by a factor of between 50x and 100x.

Figure 3 shows the number of false positives at healthy members (`FP- Events`) for each number of concurrent anomalies tested. It is more noisy, compared to Figure 2, due to these events being much less frequent than false positives in general. Once again, the number of false positives rises with the number of concurrent anomalies, and at every concurrency level, full Lifeguard (`Lifeguard`) reduces the number of false positives at healthy members by a factor of between 10x and 100x. The false positive rate is reduced so much with Lifeguard fully enabled that at some concurrencies, zero false positives occurred at healthy nodes during repeated testing.

2) Detection and Dissemination Latency: The Threshold experiment, described in Section V-D1, is used to measure the effect of Lifeguard on detection and dissemination latency for true positive failures.

Table V shows the effect of the different Lifeguard components on detection and dissemination latencies across all experiments where $\alpha = 5$ and $\beta = 6$. The meaning of each column is as follows:

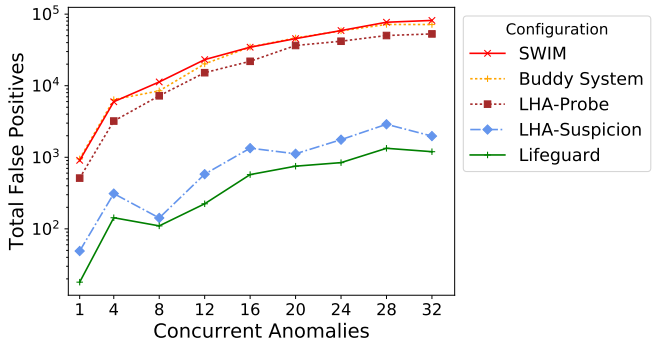

Fig. 2. **Total false positives (`FP Events`) versus number of concurrent anomalies for all experiments where $\alpha = 5$ and $\beta = 6$.**

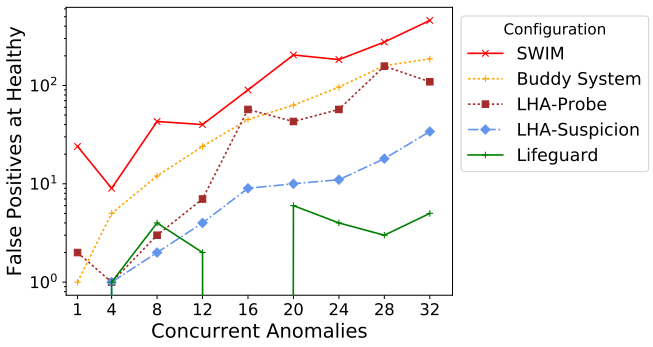

Fig. 3. **False positives at healthy agents (`FP- Events`) versus number of concurrent anomalies for all experiments where $\alpha = 5$ and $\beta = 6$.**

Configuration Tested	Median 1st Detect	99th % 1st Detect	99.9th % 1st Detect	Median Full Dissem	99th % Full Dissem	99.9th % Full Dissem
SWIM	12.44	16.96	19.40	12.90	16.93	20.17
LHA-Probe	12.42	17.75	20.10	12.90	17.98	20.56
LHA-Suspicion	12.42	17.47	25.41	12.89	17.33	23.80
Buddy System	12.45	17.12	19.16	12.92	17.18	19.81
Lifeguard	12.45	17.90	21.20	12.91	18.05	21.68

TABLE V
FIRST DETECTION AND FULL DISSEMINATION LATENCIES FOR ALL EXPERIMENTS WHERE $\alpha = 5$ AND $\beta = 6$. ALL TIMES ARE IN SECONDS.

- `Configuration Tested` : Combination of Lifeguard components enabled, as defined in Section V-B.
- `Median 1st Detect` : The median time from the start of an anomaly to its first detection by one other agent.
- `99th % 1st Detect` : The 99th percentile time from the start of an anomaly to its first detection by one other agent.
- `99.9th % 1st Detect` : The 99.9th percentile time from the start of an anomaly to its first detection by one other agent.
- `Median Full Dissem` : The median time from the start of an anomaly to dissemination of the failure to all healthy agents.
- `99th % Full Dissem` : The 99th percentile time from the start of an anomaly to dissemination of the failure to all healthy agents.
- `99.9th % Full Dissem` : The 99.9th percentile time from the start of an anomaly to dissemination of the failure to all healthy agents.

Table V shows that full Lifeguard (`Lifeguard`) raises the latencies for first detection and full dissemination by a small amount. The median increases 0.1 seconds (less than 0.1%) for both first detection and full dissemination. The increases in 99th and 99.9th percentile latencies are larger, at around 1 second (6-7%) for first detection and 1.5-1.8 seconds (7-9%) for 99.9th percentile. Local Health Aware Probe(`LHA-Probe`) appears to make the largest contribution to the increase in 99th percentile latencies, while Local Health Aware Suspicion (`LHA-Suspicion`) makes the largest contribution to the increases in 99.9th percentile latencies.

3) Message Load: The Interval experiment, described in Section V-D2, is used to measure the effect of Lifeguard on message load. The number of messages and total bytes sent in each experiment are captured using Consul's telemetry [14].

Table VI gives the aggregated message load statistics for all experiments where $\alpha = 5$ and $\beta = 6$. The meaning of each column is as follows:

- `Configuration Tested` : The Lifeguard components enabled, as defined in Section V-B.
- `Msgs Sent(M)` : The total number of (compound) SWIM-related messages sent by all Consul agents, in millions. Compound messages made by piggybacking gossip messages on ping-related messages are counted as one message.
- `Bytes Sent(GiB)` : The total size of the sent messages, in gibibytes.
- `Msgs % SWIM` : `Msgs Sent` as a percentage of the value for `SWIM` (the baseline).
- `Bytes % SWIM` : `Bytes Sent` as a percentage of the value for `SWIM` (the baseline).

Table VI shows that for experiments with $\alpha = 5$ and $\beta = 6$, Lifeguard leads to an average increase of around 11% in the number of messages sent, but the amount of data sent actually decreases by around 2%. Local Health Aware Suspicion (`LHA-Suspicion`) is the main contributor to the increase in both the number of messages and bytes sent. However, this effect is offset by Local Health Aware Probe (`LHA-Probe`), which reduces both the number of messages and bytes sent.

Configuration Tested	Msgs Sent(M)	Bytes Sent(GiB)	Msgs % SWIM	Bytes % SWIM
SWIM	435.33	149.15	100.00	100.00
LHA-Probe	428.62	134.28	98.46	90.03
LHA-Suspicion	484.55	158.87	111.31	106.52
Buddy System	435.62	147.67	100.07	99.01
Lifeguard	481.42	146.13	110.59	97.97

TABLE VI
AGGREGATED MESSAGE LOAD RESULTS FOR ALL EXPERIMENTS WHERE $\alpha = 5$ AND $\beta = 6$. FOR EACH CONFIGURATION TESTED, MSGS SENT IS THE TOTAL NUMBER OF (COMPOUND) MESSAGES SENT IN MILLIONS AND BYTES SENT IS THE TOTAL BYTES SENT IN GIBIBYTES. MSGS%SWIM AND BYTES%SWIM SHOW THE SAME RESULTS AS THE PERCENTAGE OF THEIR RESPECTIVE VALUES FOR THE SWIM BASELINE.

4) Suspicion Timeout Tuning: The results in the previous sections were obtained with the tunable Suspicion timeout parameters set to $\alpha = 5$ and $\beta = 6$, which are the highest values considered. We now examine the effect of lowering α and β.

Table VII shows the values for the metrics defined in Sections V-F1 and V-F2, for Lifeguard (`Lifeguard`) when configured with different combinations of α and β. The metrics are shown as a percentage of their baseline values from running the same set of experiments for SWIM (`SWIM`).

The following relationships are observed:

- All six latency measures (`Med First`, `Med Full`, `99% First`, `99% Full`, `99.9% First` and `99.9% Full`) are positively correlated with α.
- When $\alpha = 2$, the latency measures (and in particular the `99%`, and `99.9%` measures) are also positively correlated with β. The same correlation is not obvious at higher values of α.
- Total false positives (`FP`) and false positives at healthy members (`FP-`) are negatively correlated with α and β.

As a result, α and β may be used to tune the detection and dissemination latencies, at the same time as the false positive rate. Because lower values of α and β improve latency while making the false positive rate worse, a reduction in detection latency must be traded for a higher false positive rate.

However, even in the case of the most extreme trade-off ($\alpha = 2$ and $\beta = 2$), where median detection and dissemination latency are reduced by around 45% compared to `SWIM`, the false positive rate at healthy nodes `FP-` is still reduced by 68% (a 3x reduction) compared to the `SWIM` value. At the other extreme ($\alpha = 5$ and $\beta = 6$), median latencies remain at their `SWIM` levels, but false positives are reduced by over 98% (more than 50x), with modest increases in 99th and 99.9th percentile latencies.

Selecting values for α and β in between these extremes allows the trade-off between reduced latency and false positives to be tuned, albeit in a coarse-grained manner. We expose α and β as parameters of Lifeguard.

VI. RELATED WORK

To our knowledge, Lifeguard is the first work to address SWIM's sensitivity to slow message processing by the failure detector module, and possibly the first to address slow message processing by the local failure detector module of any distributed failure detector system.

We consider Lifeguard's relationship to both the literature of adaptive failure detectors and adaptive gossip protocols. We restrict the discussion to unreliable failure detection, since protocol with strong membership guarantees to not have the scaling characteristics required in the datacenter-scale setting[15].

Chandra and Toueg [16] introduce the concept of unreliable failure detectors, which is the category that encompasses SWIM, Lifeguard and most failure detectors deployed on commodity hardware. They identify completeness and accuracy as key properties for evaluating unreliable failure detectors. However, their focus is on understanding the conditions under which unreliable failure detectors can be used to build a reliable, distributed consensus protocol (or equivalently, an atomic broadcast protocol). Consequently, they only reason about failure detectors abstractly, and do not explore how a detector might be made more reliable.

Chen et al. [17][18] observe that [16] only offers eventual guarantees, with no timing assumptions. To address this, they introduce quality of service (QoS) for failure detectors, and identify detection latency as a critical property in many use cases. They propose an adaptive failure detector that, like Lifeguard, adjusts its timeouts based on recent observations about message loss and delay. But unlike Lifeguard, there is no consideration of whether the local failure detector might be running slowly, and hence a slow detector could report false positives about the peer member it is monitoring.

Bertier et al. [19] refine [17] with a better estimate of network latency, resulting in lower average detection time. Hayashibara et al. [20] make a more significant modification to [17], and introduce accrual failure detectors, which replace the traditional boolean detector output with a suspicion value on a continuous scale. This allows applications to make more nuanced decisions about the health of a monitored member. Satzger et al. [21] make the accrual detector more computationally efficient and remove the assumption of normally distributed arrival times. Most recently, Liu et al. [22] argue for a specific arrival time distribution that is better suited to the message delays seen in cloud environments. However, once again, none of these designs consider whether the local failure

	$\alpha=2$ $\beta=2$	$\alpha=2$ $\beta=4$	$\alpha=2$ $\beta=6$	$\alpha=4$ $\beta=2$	$\alpha=4$ $\beta=4$	$\alpha=4$ $\beta=6$	$\alpha=5$ $\beta=2$	$\alpha=5$ $\beta=4$	$\alpha=5$ $\beta=6$
Med First	53.14	54.10	54.34	82.96	83.04	83.12	99.76	99.52	100.08
Med Full	55.12	56.28	56.74	84.42	84.03	84.42	99.92	99.61	100.08
99% First	69.81	72.88	75.53	94.28	96.17	96.82	104.95	102.71	105.54
99% Full	73.07	76.96	79.15	97.05	96.69	96.52	105.73	105.08	106.62
99.9% First	76.08	75.41	80.36	99.07	93.71	94.69	112.32	111.44	109.28
99.9% Full	76.20	75.11	78.58	92.17	95.14	92.71	107.64	107.93	107.49
FP	98.37	43.64	24.16	37.72	8.04	3.18	26.61	5.43	1.53
FP-	31.15	22.47	13.65	20.29	9.50	4.83	15.38	5.05	1.89

TABLE VII

PERFORMANCE AS PERCENTAGE OF SWIM BASELINE WITH DIFFERENT TUNINGS OF α AND β. EACH COLUMN SHOWS METRICS FOR LIFEGUARD CONFIGURED WITH THE GIVEN VALUES OF α AND β. THE METRICS ARE THOSE DEFINED IN SECTIONS V-F1 AND V-F2, SHOWN AS A PERCENTAGE OF THEIR BASELINE VALUES FOR SWIM (SWIM).

detector is running slowly, and hence they all have the same potential as [17] for a slow running detector to make false positive reports about a healthy peer that it is monitoring.

All of the above work is heartbeat-based. We observe that there is nothing inherent to heartbeats that prohibits modeling of the local detector's timeliness. However all of these works focus on the operation of a single failure detector in a 1-to-1 monitoring relationship with a single peer, which means the Lifeguard heuristics can not be applied directly. In the setting of multiple co-located heartbeat-based detectors (each receiving messages from a different peer), it would be possible to evaluate applying the Lifeguard heuristics. We return to this point in Section VII.

Gupta et al. [23] introduce adaptivity into the gossip literature. Like Lifeguard, their adaptive scheme leverages local knowledge about peer failure and message loss, and uses it to take remedial action (in their case to transition to a different dissemination sub-protocol). However, unlike Lifeguard, there is no consideration of slowness, either of message delivery or members themselves. Additionally, the metrics evaluated are instantaneous, rather than accumulated over a period of time, and do not take into account correlation across different peers.

A number of other adaptive gossip protocols are similar to Lifeguard in that they adjust the sending of messages, based on the local member's interaction with its peers. Levis et al. [24] and Gobriel et al. [25] delay forwarding messages, while Haas et al. [26][27] and Kyasanur et al. [28] vary the probability of forwarding a message. Bhandari and Gupta [29] vary both forwarding delay and probability. However, these protocols all target Wireless Sensor Networks (WSNs), and their adaptive behavior is concerned with eliminating unnecessary messages. They adapt passively to member failures, and do not use probe-based failure detection or offer timeliness guarantees. Hence they have no need to model slow message processing.

Johansen et al. [30] propose an adaptive gossip-based protocol that is similar to Lifeguard in many respects. Like SWIM and hence Lifeguard, it is a general purpose group membership protocol, which uses probe-based failure detection. It has a suspicion phase and gossip based update dissemination sub-protocol. Like Lifeguard, it adaptively tunes the probe timeouts, but it is more granular, with an independent tuning for each peer that is probed. (This is possible because the set of probe targets is based on membership in the same pseudo-random ring, and hence is very small and stable compared to that of SWIM and Lifeguard, which round robin through all known peers.) However, unlike Lifeguard, the suspicion time-out it not adaptively tuned. More significantly, the probe tuning does not consider the possibility of slow message processing at the local failure detector. In fact, the assumption that all correct (meaning non-Byzantine and non-failed) members can run their probe and update dissemination sub-protocols in a timely manner is explicitly state (in section 4.1). The adaptive tuning is present only to accommodate unreliable message delivery.

VII. CONCLUSIONS AND FUTURE WORK

Our goal with Lifeguard was to reduce the rate of false positive failures compared to that of SWIM, while minimally impacting latencies and message load. Across a wide range of cases tested, Lifeguard achieves this, with reductions in false positives in the range of 10x to 100x, and over 50x on average. The false positive rate is reduced so much that at some levels of concurrent anomalies, zero false positives occurred at healthy nodes during repeated testing. This is achieved with negligible increase in median detection and dissemination latencies, and modest (6-9%) increase in 99 and 99.9th percentile latencies. On average, around 12% more messages are sent, but the total bytes sent actually falls around 2%.

Additionally, through tuning of the timeouts used by Lifeguard's Local Health Aware Suspicion component, some of the reduction in false positives can be traded for a reduction in latencies. But even in the case of the most extreme trade-off tested, where median detection and dissemination latency are reduced by 45% (close to 2x), the false positive rate at healthy nodes is still reduced by 68% (3x), compared with SWIM.

All measures of detection and dissemination latency are reduced by the tuning, however the gap between median and 99th percentile latencies widens as the median latency is decreased. This is not surprising, given that Lifeguard's selection of peers to communicate with, like SWIM's, is randomized and has no coordination between members. Future

work could explore ways to more tightly bound detection and dissemination latencies. Adding a random overlay network is one possible approach, and in particular we look to [31] for inspiration.

Lifeguard has several parameters that currently use heuristically determined values. These include Local Health Aware Suspicion's re-gossip factor (K), the saturation limit of the LHM counter (S) and the scores given to the different events that affect the LHM counter. Future work could explore automatic tuning of these parameters, with one possible approach being to find (or learn) metrics that allow these parameters to be adaptively tuned via feedback.

In developing Lifeguard, we have devised heuristics that take advantage of the randomized patterns of communication that Lifeguard inherits from SWIM. Future work could replace Lifeguard's heuristics with a formal model or new heuristics derived from a model as in [31], or from a utility function, as in [32]. A separate line of work could investigate applying the local health approach to other classes of failure detector.

REFERENCES

[1] A. Das, I. Gupta, and A. Motivala, "SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, ser. DSN '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 303–312.

[2] (2017, Nov.) Butterfly documentation. Chef. [Online]. Available: https://www.habitat.sh/docs/internals/#supervisor-internals

[3] (2017, Nov.) Habitat. Chef. [Online]. Available: https://www.habitat.sh

[4] (2017, Nov.) Ringpop documentation. Uber Technologies Inc. [Online]. Available: https://ringpop.readthedocs.io/en/latest/architecture_design.html

[5] HashiCorp, "memberlist Project," https://github.com/hashicorp/memberlist, 2017, [Online; accessed 28-Feb-2017].

[6] ——, "Consul Project," https://www.consul.io/, 2017, [Online; accessed 28-Feb-2017].

[7] ——, "Nomad Project," https://www.nomadproject.io/, 2017, [Online; accessed 28-Feb-2017].

[8] ——, "Consul vs. Serf," https://www.consul.io/intro/vs/serf.html, 2017, [Online; accessed 28-Feb-2017].

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643666

[10] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," in *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, 2001, pp. 170–179. [Online]. Available: http://doi.acm.org/10.1145/383962.384010

[11] HashiCorp, "Serf Project," https://www.serf.io/, 2017, [Online; accessed 28-Feb-2017].

[12] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '87. New York, NY, USA: ACM, 1987, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/41840.41841

[13] T. Bowden, B. Bauer, J. Nerin, S. Feng, and S. Seibold, "The /proc Filesystem," https://www.kernel.org/doc/Documentation/filesystems/proc.txt, 2009, [Online; accessed 28-Feb-2017].

[14] HashiCorp, "Consul Telemetry," https://www.consul.io/docs/agent/telemetry.html, 2017, [Online; accessed 28-Feb-2017].

[15] I. Gupta, K. P. Birman, and R. van Renesse, "Fighting fire with fire: using randomized gossip to combat stochastic scalability limits," *Quality and Reliability Engineering International*, vol. 18, no. 3, pp. 165–184, 2002.

[16] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, Mar. 1996.

[17] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," in *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, 2000, pp. 191–200.

[18] ——, "On the quality of service of failure detectors," *IEEE Trans. Computers*, vol. 51, no. 1, pp. 13–32, 2002.

[19] M. Bertier, O. Marin, and P. Sens, "Implementation and performance evaluation of an adaptable failure detector," in *Proceedings International Conference on Dependable Systems and Networks*, June 2002, pp. 354–363.

[20] N. Hayashibara, X. Defago, R. Yared, and T. Katayama, "The phi; accrual failure detector," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, Oct 2004, pp. 66–78.

[21] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A new adaptive accrual failure detector for dependable distributed systems," in *Proceedings of the 2007 ACM Symposium on Applied Computing*, ser. SAC '07. New York, NY, USA: ACM, 2007, pp. 551–555.

[22] J. Liu, Z. Wu, J. Wu, J. Dong, Y. Zhao, and D. Wen, "A weibull distribution accrual failure detector for cloud computing," *PLOS ONE*, vol. 12, no. 3, pp. 1–16, 03 2017.

[23] I. Gupta, A. M. Kermarrec, and A. J. Ganesh, "Efficient epidemic-style protocols for reliable and scalable multicast," in *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, 2002, pp. 180–189.

[24] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI'04. Berkeley, CA, USA: USENIX Association, 2004.

[25] S. Gobriel, D. Mosse, and R. Melhem, "Mitigating the FloodingWaves Problem in Energy-Efficient Routing for MANETs," in *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, 2006, pp. 47–47.

[26] Z. J. Haas, J. Y. Halpern, and L. Li, "Gossip-based ad hoc routing," in *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1707–1716 vol.3.

[27] ——, "Gossip-based ad hoc routing," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 479–491, June 2006.

[28] P. Kyasanur, R. R. Choudhury, and I. Gupta, "Smart gossip: An adaptive gossip-based broadcasting service for sensor networks," in *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Oct 2006, pp. 91–100.

[29] V. Bhandari and I. Gupta, "Prioritycast: Efficient and time-critical decision making in first responder ad-hoc networks," in *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Oct 2006, pp. 246–255.

[30] H. Johansen, A. Allavena, and R. van Renesse, "Fireflies: Scalable support for intrusion-tolerant network overlays," in *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, ser. EuroSys '06. New York, NY, USA: ACM, 2006, pp. 3–13.

[31] J. A. Patel, I. Gupta, and N. Contractor, "Jetstream: Achieving predictable gossip dissemination by leveraging social network principles," in *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, July 2006, pp. 32–39.

[32] G. He, R. Zheng, I. Gupta, and L. Sha, "A framework for time indexing in sensor networks," *ACM Trans. Sen. Netw.*, vol. 1, no. 1, pp. 101–133, Aug. 2005.