Medley: A Membership Service for IoT Networks

Rui Yang, Jiangran Wang, Jiyu Hu, Shichu Zhu, Yifei Li, and Indranil Gupta

***Abstract*—Efficient and correct operation of an IoT network requires the presence of a failure detector and membership protocol amongst the IoT nodes. This paper presents a new failure detector for IoT settings wherein nodes are connected via a wireless ad-hoc network. Our failure detector, named *Medley*, is fully decentralized, allows IoT nodes to maintain a local membership list of other alive nodes, detects failures quickly (and updates the membership list), and incurs low communication overhead. We adapt a failure detector originally proposed for datacenters (SWIM), for the IoT environment. This adaptation is non-trivial. In Medley each node picks a medley of ping targets in a randomized and skewed manner, preferring nearer nodes. We also provide optimizations to achieve time-bounded detection, as well as to reduce tail detection times. Via analysis, simulation, and Raspberry Pi deployments, we show that Medley can simultaneously optimize detection time and communication traffic.**

***Index Terms*—Failure detection, Internet of Things, Membership**

I. Introduction

The IoT market is expected to reach 500 Billion dollars in size by 2022 [1]. For instance, during just the second quarter of 2018, Amazon Echo + Dot sold 3.6 million units, while Google Home + Mini sales were 3.1 million units [2]. IoT deployments in smart buildings, smart homes, smart hospitals, smart forests, battlefield scenarios, etc., are proliferating. While today's deployments in smart homes are typically a few tens of devices, tomorrow's vision, in smart buildings and cities, is for hundreds or thousands of devices communicating with each other.

Such large IoT deployments are in essence *distributed systems* of devices. As such, there is a need to provide familiar abstractions and a similar substrate of distributed group operations as those which exist in internet-based distributed systems like datacenters, peer-to-peer systems, clouds, etc. In other words, a *distributed group communication substrate* is required for IoT settings, atop which management functions and distributed programs can then be built. This is critical in order to build large-scale IoT deployments that are truly autonomous, self-healing, and self-sufficient.

One of the first problems that such a substrate needs to solve is detecting failures (we consider only fail-stop failures in this paper[1]). At large scale, failures are the norm rather than the exception. When a device fails, other affected devices need to know about it and take appropriately corrective action, and in some cases inform the human user. This is a very common way of building internet-based and datacenter-based distributed systems. In the IoT environment, examples of corrective actions after failure include (but are not limited to): backup actions to ensure user needs are met (e.g., maintain sufficient lighting in an area), re-initiating and re-replicating device schedules that were stored on failed devices (e.g., timed schedules), informing the upper management layer, informing the user, etc.

Existing techniques in IoT literature detect failures either centralized or semi-centralized [3], [4], [5], [6]. These typically provide a central clearinghouse where information is maintained about currently-alive nodes. Yet, they require access to a cloud or a cloudlet, but this is not always feasible. For instance, IoT deployments may span remote scenarios (e.g., battlefields, forests, etc.), and in some cases sending data to the cloud may be prohibited by laws (e.g., GDPR [7] or HIPAA [8] laws for data from smart hospitals). Additionally, if the centralized service becomes inaccessible (e.g., to due to failures or message losses), the IoT devices no longer have access to the failure detection and membership service.

In this paper, we present *Medley*, which is the first fully-decentralized membership service for IoT distributed systems running over a wireless ad-hoc network. The Medley membership service maintains at each IoT node, a dynamic membership list containing a list of currently alive nodes in the system. The membership service's critical goal is to detect device failures (crashes) and update membership lists at non-faulty nodes—this is the responsibility of the *failure detector* component, which is the focus of this paper. Like other practical membership systems [9], Medley is also weakly-consistent membership service: membership changes (failures, joins, leaves) propagate eventually. We measure how quickly they propagate, and how much bandwidth they consume.

Maintaining full membership lists at devices is scalable and feasible in an IoT setting. Consider a system with 5K devices (sufficient to densely populate a multi-storey office building). Suppose each membership list entry uses 20 B (16 B for IPv6 address + 4 B for sequence number). This entails 100 KB of memory for the membership list. For Raspberry Pis which currently have such as 2G of memory, the membership list would occupy only 0.005% of the memory.

Classical distributed systems literature builds a wide swath of distributed algorithms over a full membership list (at each node). Examples include multicast, coordination, leader election, mutual exclusion, virtual synchrony etc. [10]. Essentially, full membership offers maximal flexibility in designing arbitrary distributed algorithms on top of it. It also helps make analysis tractable. For IoT networks, Medley opens the door for similar algorithms to be built on top of it. For instance, to build a multicast tree, one algorithm could choose only nearby nodes, or alternatively a mix of near and far nodes. Both can be built atop a full membership algorithm.

Failure detector protocols for internet-based distributed systems fall into two categories: heartbeat-based (or lease-based), and ping-based. Heartbeat-based protocols [11], [12], [13]

[1]Malicious/Byzantine failures are outside our scope.

have each node send periodic heartbeats to one or more other monitor nodes; when a node n_i dies, its heartbeats stop, the monitors time out, and detect the node n_i as failed. Ping-based protocols [14], [9] have each node periodically ping randomly-selected target nodes from the system. Analysis in [14] has shown that compared to heartbeat protocols, ping-based protocols are faster at detecting failures and impose less network traffic, and can completely detect failures.

We thus adopt a ping-based approach for our IoT failure detection protocol Medley. The key challenge for Medley is that existing ping-based protocols [9], [15] select ping targets uniformly at random across the system. Randomized selection is attractive due to its fast detection, congestion avoidance and load balancing. Yet in a wireless ad-hoc IoT network, uniform random selection leads to large volumes of network traffic that span major portions of the IoT network.

Medley solves this by proposing a new *spatial* ping-target selection strategy which prefers nearer nodes but also has some probability of pinging farther nodes. Compared to fully randomized pinging, always picking nearby nodes as ping targets localizes and reduces network traffic. But this always-local selection leads to high detection times due to lowered randomness of pinging. It also causes non-detection of failures when multiple simultaneous failures occur (e.g., failures caused by a circuit breaker tripping), because all nearby pingers of a failed node have also failed.

Medley attempts to gain the advantages of both approaches by using a hybrid of the uniform-random and the always-local target selection. It utilizes a mix (medley) of nearer and farther ping targets. A key question we answer both analytically and empirically is: What is the best way to mix these targets? We also present two optimizations to reduce tail latency of detection time.

The contributions of this paper are:

1) A new fully-decentralized failure detector protocol, named *Medley*, for wireless ad-hoc IoT networks.
2) Mathematical analysis of the key parameter (exponent) in spatial pinging, in order to optimize detection time as well as communication traffic.
3) An optimization to provide time-bounded detection of failures in Medley.
4) Two optimizations to reduce tail detection times.
5) Evaluation of Medley via simulations in Java (matching deployment) and NS-3 (for link layer fidelity).
6) Implementation of Medley for Raspberry Pi, and subsequent deployment experiments.
7) Compared to classical techniques, Medley provides comparable failure detection times, lowers bandwidth by 37.8% (given a detection time), and false positive rates of 2% under high 20% packet drop rates. We cut tail detection time by up to 47.2%.

II. Background

System Model: We consider the fail-stop model: once a node crashes it executes no further instructions or operations. Fail-recovery models can be seen as a special case (with nodes rejoining under a new id or incarnation number). Byzantine failures [16] are beyond our current scope (but represent an interesting future direction).

The network is asynchronous, and messages may be delayed or dropped. Multiple nodes may fail simultaneously. Nodes are allowed to join and voluntarily leave the system. We use N to denote the number of nodes in the system.

Each node maintains a membership list consisting of entries for all other nodes in the system–our membership protocol's goal is to delete entries for failed/departed nodes soon after their failure departure, and to add entries for joining nodes soon after they join. Our protocol makes no assumptions about clock synchronization, but our analysis assumes (for tractability) that clock speeds are similar.

Failure Detector Properties: Failure detectors have three desirable properties. The two desirable correctness properties are called [17] *Completeness* and *Accuracy*. Completeness requires that every failure is detected by at least one non-faulty node. Accuracy means that no failure detections are about healthy nodes, i.e., there are no false positives. In their seminal paper [17] Chandra and Toueg proved that it is impossible to design a failure detector for asynchronous networks, to satisfy both completeness and accuracy. Due to the need to perform corrective recovery actions after a failure, today's failures navigate this impossibility by always guaranteeing completeness, while attempting to maximize accuracy (i.e., minimize false positive rate).

Besides the above two properties, failure detectors also aim to minimize *detection time*, i.e., time between failure and first non-faulty node discovering this failure. Finally, scalability and load balancing are often goals of failure detectors.

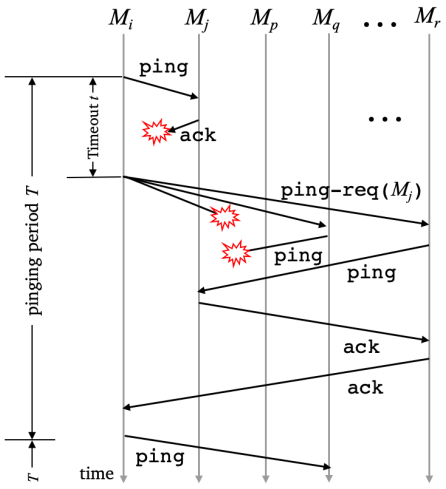


Fig. 1: **One pinging round in SWIM failure detection [14].**

SWIM Failure Detector: Our Medley system is adapted from the failure detector and dissemination component of the SWIM protocol [9], [14]. SWIM is popular and various versions of it are today widely deployed in datacenters and in open-source software, including at Uber [18], and HashiCorp's Serf [19] and Consul [20].

We next describe the base SWIM protocol to set the context for Medley. The SWIM membership protocol handles failure detection and dissemination separately. The former detects

failures, while the latter multicasts to the system information about node joins, leaves, and detected failures.

Fig. 1 (from [9], [14]) depicts the SWIM failure detector. Each node M_i periodically runs the following protocol every T time units. T is fixed at all nodes but nodes run their periods asynchronously from each other. Each period consists of a *direct pinging* phase and an optional *indirect pinging* phase.

At the start of a period, M_i picks a member from its membership list, uniformly at random, and sends it a ping message. Any node M_j receiving a ping responds immediately with an ack. If M_i receives the ack within a small timeout t (based on message RTT), then M_i is satisfied and does nothing else in this period. Otherwise, M_i picks k other nodes (denoted as indirect pingers), also at random, and asks each of them to ping M_j. If any of these k nodes hears back an ack from M_j, they pass on the ack back to M_i. If M_i receives at least one such ack before the end of the period, it is satisfied and does nothing else in this period. Otherwise, i.e., if M_i hears no acks, then it marks M_j as failed at the end of this period. Pings and acks carry unique identifiers to avoid confusion with other rounds and pingers.

Indirect pinging essentially gives a "second chance" to pinged nodes that might have been congested or slow during the initial ping. It also avoids potential network congestion on the direct $M_i - M_j$ network path. Both of these reduce false positive rates.

Analysis in [14] shows that even without the indirect pinging, failures are detected within $O(1)$ protocol periods on expectation. In addition, the SWIM protocol guarantees eventual detection of all failures (eventual completeness).

SWIM Dissemination Component: SWIM nodes continuously piggyback the information about node join/leave/failure atop the messages they send out, namely ping, ack, and indirect ping request for quick dissemination. In addition, a receiving node records new information in the message and reacts accordingly.

This "infection-style" dissemination provides a gossip-like behavior for disseminating information about node failures, joins, and voluntary leaves. Analysis [14] shows that in a system with N nodes, information spreads with high probability to all nodes within $O(log(N))$ time periods.

III. Medley: Design and Analysis

A. Spatial Pinging

We target settings where IoT devices are connected via a wireless ad-hoc network. In such scenarios, the SWIM failure detector described in Sec. II is inefficient because it picks ping targets uniformly at random. This spreads pings and acks across far distances in the ad-hoc network. Far pings and acks require more routing hops, incurring higher communication overhead on intermediate nodes, longer latency, and create congestion and packet losses.

Thus, we propose in Medley a way to replace the randomized target selection in SWIM with a *skewed randomized* mechanism which takes distance to target into account. We call this as *spatial target selection*.

Spatial Target Selection: In Medley, a node chooses to ping a given target with probability proportional to $\frac{1}{r^m}$, where r is the distance to the target and m is a fixed exponent.

An example is shown in Fig. 2. M_i has in its membership list nodes M_p, M_q, and M_r at distances $d, 2d,$ and $4d$ respectively. In a period of the SWIM protocol at M_i, it has the highest probability ($\propto \frac{1}{d^m}$) of pinging M_p. Similarly, the probabilities for pinging M_q, M_r are respectively $\propto \frac{1}{(2d)^m}$, and $\propto \frac{1}{(4d)^m}$. Using appropriate normalization constants, we depict two points in the space of m. If $m = 1$, then the respective ping probabilities to M_p, M_q, M_r are 0.57, 0.28, and 0.15. However, increasing the exponent m to 2 localizes pings more–the changed ping probabilities are respectively 0.75, 0.2, and 0.05. M_p with probability 0.75 will be pinged even more frequently.

The above calculations indicate that higher values of m localize ping-ack traffic more and incur lower communication overhead. At the same time, more localized pinging reduces the randomness of pinging and thus increases the detection time. We wish to find "good" values for m that optimize both network traffic and detection time. We do so in Sec. III-B.

We point out that Spatial Pinging (Medley) is a generalization of SWIM. When $m = 0$, spatial pinging degenerates to SWIM with uniform target selection. $m = \infty$ means that each member uniformly pings to its closest neighbours.

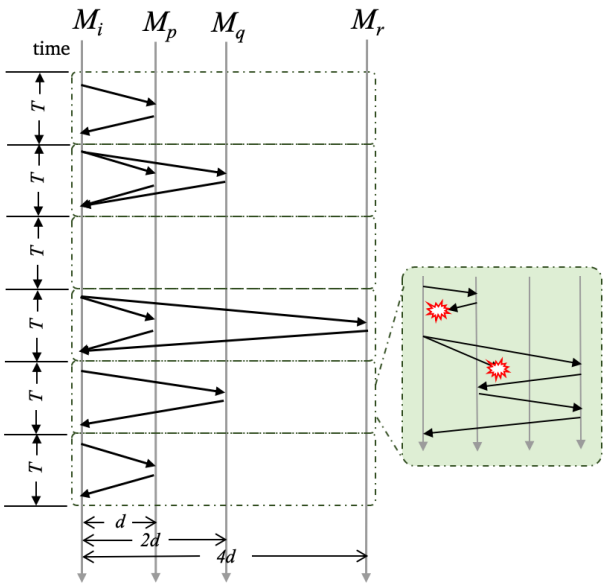


Fig. 2: **Example of ping target selection in Medley.**

Other components: Just like SWIM, Medley disseminates information by piggybacking atop pings, acks, and indirect pings (Sec. II, "SWIM Dissemination Component"). This is a gossip style of dissemination and is also used to disseminate node join/leave information.

Medley is able to seamlessly borrow optimizations from SWIM. One such important optimization is suspicion, which allows mistakenly-detected alive nodes a second chance to disprove their false detection. Here a detected node is not marked as failed but instead is suspected and this suspicion gossiped to other nodes (via pings and acks). If another node

successfully pings the suspected node via normal pinging, before the suspicion times out, the suspected node rejuvenates in membership lists and is not deleted from membership lists. More details can be found in the SWIM paper [9].

B. Analysis

We analyze Medley's spatial pinging under certain idealized assumptions. For tractability, we assume that: i) the N nodes are uniformly spread with a density of D, and ii) a pinging node picks targets only up to a distance of R away.

First, to minimize detection time we wish to maximize the expected number of pings a given node receives during a pinging period. We denote this expected number as E[Pings received per period] or $EP(m)$, where:

$$\begin{aligned} EP(m) &= (\int_0^R \frac{1}{r^m} D(2\pi r) \cdot dr) \cdot \frac{1}{\pi R^2 D} \\ &= (\int_0^R \frac{1}{r^{m-1}} \cdot dr) \cdot \frac{2}{R^2} \end{aligned} \tag{1}$$

In the first line of the equation, the integral term contains the probability of being picked as a ping target($\frac{1}{r^m}$), multiplied by the number of nodes in an annulus at radius r ($D(2\pi r) \cdot dr$). The term beyond parentheses is a normalizing constant to ensure that when $m = 0$, which is the uniform default SWIM, Equation 1 comes to an expected 1 received ping.

Second (along with maximizing ping probability), we simultaneously wish to minimize communication cost $C(m)$ incurred by pings received at a given node. A message transits over multiple hops in the underlying ad-hoc network. Assuming a fixed size for messages, $C(m)$ is proportional to the number of hops incurred by the message. Again for tractability, we calculate a message's cost as proportional to the distance between its sender and receiver (as this is correlated with hop count). We obtain:

$$\begin{aligned} C(m) &= (\int_0^R \frac{1}{r^m} D(2\pi r) \cdot r \cdot dr) \\ &= (\int_0^R \frac{1}{r^{m-2}} \cdot dr) \cdot (2\pi D) \end{aligned} \tag{2}$$

This is obtained by multiplying the expected number of pings in the annulus of radius r (similar to Equation 1), by the communication cost incurred by the multi-hop network, which is proportional to the target distance r.

In order to simultaneously minimize $C(m)$) and maximize $EP(m)$, we define our optimization function that we wish to maximize as: $Ratio(m) = \frac{EP(m)}{C(m)}$.

Theorem 1. *Medley's spatial pinging: (i) provides completeness, and (ii) optimizes $Ratio(m)$ at the following values of exponent m:*

1) *If the ratio of deployment area dimension to inter-node distance is high, then $m = \infty$ is optimal;*
2) *If the ratio of deployment area dimension to inter-node distance is low, then $m = 3$ is optimal.*

Proof. First, to prove completeness, consider a failure of node M_j. We observe that with at least one non-faulty node M_i in the system, M_i has a non-zero probability of pinging M_j during any protocol period subsequennt to M_j's failure. Because of the (biased) randomness of picking ping targets, M_i is guaranteed to *eventually* pick M_j as a ping target in a future period. M_j will be unresponsive (because it is failed), and thus M_i will mark M_j as failed.

Second we find the optimal value of m. Explanding the expected ping count $EP(m)$ gives us:

$$EP(m) = \begin{cases} \frac{R^{2-m}}{2-m} \cdot \frac{2}{R^2} & m < 2 \\ \log(\frac{R}{d}) \cdot \frac{2}{R^2} & m = 2 \\ (\frac{1}{d^{m-2}} - \frac{1}{R^{m-2}}) \cdot \frac{1}{m-2} \cdot \frac{2}{R^2} & m > 2 \end{cases} \tag{3}$$

where d represents the distance to the nearest node (for a 2-dimensional deployment, $d \propto \frac{1}{\sqrt{D}}$).
Similarly, for communication cost $C(m)$:

$$C(m) = \begin{cases} \frac{R^{3-m}}{2-m} \cdot (2\pi D) & \text{when } m < 3 \\ \log(\frac{R}{d}) \cdot (2\pi D) & \text{when } m = 3 \\ (\frac{1}{d^{m-3}} - \frac{1}{R^{m-3}}) \cdot \frac{1}{m-3} & \text{when } m > 3 \end{cases} \tag{4}$$

Table I shows the variation of $Ratio(m) = \frac{EP(m)}{C(m)}$ as m is increased (second column). To make comparisons tractable, the third column shows the normalized value $\frac{Ratio(\infty)}{Ratio(m)}$. We wish to minimize this value (in order to maximize $Ratio(m)$).

TABLE I: **Ratio of expected number of pings (need to maximize) to Communication (need to minimize). Here $x = \frac{R}{d}$. Constants elided.**

m	$Ratio(m)$	$\frac{Ratio(\infty)}{Ratio(m)}$
0	$\frac{3}{R^3}$	$\frac{2}{3} \cdot x$
1	$\frac{4}{R^3}$	$\frac{1}{2} \cdot x$
2	$\frac{2\log(\frac{R}{d})}{R^3}$	$\frac{x}{\log(x)}$
3	$\frac{2}{dR^2\log(\frac{R}{d})}$	$\log(x)$
>3	$\frac{2 \cdot (m-3)}{(m-2) \cdot dR^2}$	Increasing, tends to 1

From the table, we can eliminate some possibilities for optimizing $Ratio(m)$:

1) $m = 0$ can be ignored as $Ratio(m = 1)$ is higher than $Ratio(m = 0)$,
2) $m = 2$ can be ignored as $\frac{x}{\log(x)}$ has a minimum of e(> 1),
3) $m = 1$ can be ignored as $\frac{Ratio(3)}{Ratio(1)} = \frac{x}{2\log(x)}$ has a minimum at $\frac{e}{2} > 1$.

Therefore, the choice for optimizing $Ratio(m)$ boils down to either $m = 3$ or $m = \infty$. Next we observe that:

1) If $R \gg d$ (in particular $\frac{R}{d} > \mathrm{e} \simeq 2.718$ or $\log(\frac{R}{d}) > 1$, then $m = \infty$ is optimal. In other words, *if the dimension of the IoT installation area is much larger than inter-node distances, local pinging is optimal.*
2) If $\frac{R}{d} < \mathrm{e} \simeq 2.718$, $m = 3$ is optimal. In other words, *for small installation areas (e.g., a room or a floor, where R is small), or areas of low node density (where inter-node distance d is high), Medley with $m = 3$ is optimal.*

□

Theorem 2. *In an area with symmetric pinging (e.g., large deployment, or 3 dimensional area), when Medley is configured to have each node send 1 ping per period [2], it achieves an O(1) expected time for failure detection, while imposing an O(1) message load.*

Proof. Consider a system of N nodes $M_1, M_2, \ldots M_N$. Without loss of generality, let M_1 be the node failing. Denote as $PP_m(i)$ the probability of M_i pinging M_1 in a given period, according to the normalized spatial ping distribution and m. Because each Medley node sends 1 ping per period, by symmetry, a node M_1 will also receive an expected 1 ping per period. This means that $\sum_{k=2}^{N} PP_m(k) = 1$, for all values of spatial exponent m we may choose.

Now the probability that *at least one* of the nodes $M_2, \ldots M_N$ picks M_i as ping target in a protocol period (and thus detects its failure) is $FP(m) = 1 - \prod_{k=2}^{N}(1 - PP_m(k))$. Because the product of terms with a fixed sum ($\prod_{k=2}^{N}(1 - PP_m(k))$) is *maximized* when all terms ($PP_m(k)$) are identical, we have for all m, $FP(m) \geq FP(m = 0)$.

When $m = 0$ (the default uniform SWIM), each of the nodes $M_2, \ldots M_N$ pings M_i per period with identical probability $\frac{1}{N-1}$. Thus, $FP(0) = 1 - (1 - \frac{1}{N-1})^{N-1} \simeq 1 - e^{-1}$ for large N. This is equivalent to tossing a coin with heads probability $(1 - e^{-1})$ per period. Thus: i) the expected detection time at $m = 0$ is $O(\frac{1}{1-e^{-1}})$ periods, which is O(1); and ii) the time for the failure to be detected with high probability (w.h.p.) $(1 - \frac{1}{N})$ is $log_{(\frac{1}{1-e^{-1}})}(N)$ periods.

Since $FP(m) \geq FP(0)$, the expected detection time and w.h.p. detection time for spatial pinging are both $\leq$ the corresponding values for $m = 0$.

□

IV. Time-Bounded Failure Detection

Theorem 1 was able to prove that detection is eventual. In practice this could still mean particularly long detection times in IoT scenarios. Consider a node M_i that is "far" from most other nodes. Because ping probabilities to M_i are low, when M_i fails, the biased target selection choices imply that it may be an arbitrarily (and indeterminately) long time for the first non-faulty node to pick M_i as ping target.

We now present an optimization that preserves the biased randomness of the Medley's spatial pinging from Section III, but is additionally able to specify an absolute time bound on how long a failed node takes to be detected.

[2]Note that this is a different assumption from the analysis in Equation 3, but is closer to our real implementation.

A. Design of Time-Bounded Medley

The key idea is to ping via a round-robin mechanism which is weighted by ping probability.

Algorithm 1 Time-bounded target selection in a super round from a single node M_i's view.

Require: Runtime Probability List $\mathcal{P}_{M_i}$
▷ *Super round:* Create initial bag $\mathcal{BAG}_{M_i}$
1: **for** each p_j in $\mathcal{P}_{M_i}$ **do**
2: Put $\lceil \frac{p_j}{p_{min}} \rceil$ into $\mathcal{BAG}_{M_i}$
3: **end for**
4: Create an empty set $onePassTargets$

5: ▷ Start target selection
6: **while** $\mathcal{BAG}_{M_i}$ is not all zeros **do**
7: ▷ Initialize new *Pass* if needed
8: **if** $onePassTargets$ is empty **then**
9: $onePassTargets =$ $\{M_j\}$ for all j that $Count_j > 0$ in $\mathcal{BAG}_{M_i}$
10: **end if**
11: ▷ One *Period*
12: Randomly pick one node in $onePassTargets$ as PING target
13: Remove M_j from $onePassTargets$
14: Reduce $\mathcal{BAG}_{M_i}(M_j)$ by one
15: **end while**

Consider a member (node) M_i with membership list $\mathcal{ML}_i$, currently containing K entries $(M_1, M_2, \ldots, M_K)$. M_i also maintains a runtime probability list $\mathcal{P}_{M_i} = [p_1, p_2, p_3 \ldots p_K]$, where p_j is the pinging probability of respective member M_j from $\mathcal{ML}_i$.

The p_j values in $\mathcal{P}_{M_i}$ are calculated using the spatial ping probabilities of Sec. III. The pseudocode for our approach is shown in Algorithm 1. We explain below.

Let $p_{min} = \min\{\mathcal{P}_{M_i}\}$, the lowest probability among all non-faulty members in $\mathcal{ML}_i$. Now, denote $Count_j = \lceil \frac{p_j}{p_{min}} \rceil$. We create a initial *bag list* as:
$\mathcal{BAG}_{M_i} = [Count_1, Count_2, \ldots Count_K]$ (Line 1 - 3).

The weighted round-robin pinging at node M_i creates a *bag* B_i which consists of $Count_j$ instances of node M_j for each $M_j \in \mathcal{ML}_i$. This can be thought of as a bag of balls, with $Count_j$ balls of color M_j.

During each period, M_i picks one ball from this bag (without replacement), and uses the corresponding member as ping target for that period. The bag is created at the start of a *super round* (which consists of multiple periods), and a super round completes when the bag is empty. Thus, a super-round consists of $(\Sigma_1^K Count_j)$ number of protocol periods.

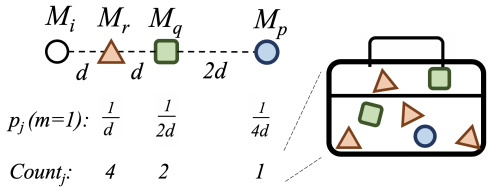
Picking these balls (targets) uniformly at random from the bag causes high variance in detection times. To reduce this, we introduce the notion of *passes*. Algorithm 1 depicts how M_i selects targets in a super round. At M_i, ping target selection is done randomly but in multiple passes through the bag. Each pass consists of multiple periods. In each pass at M_i, every node M_j (in M_i's bag), which has at least one leftover instance in the bag, is touched (removed, and pinged) only once. These instances are removed in a random order (Line 8 - Line 13).

Suppose a particular pass contains r instances (thus consisting of r protocol periods). Then during these r periods, M_i sequentially picks one instance as ping target based on the order. When the final pass is done (and no instances are left in the bag), all instances are put back in the bag, a new super round is started, and the above process is repeated.

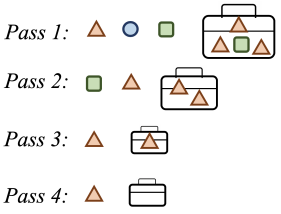
Note that the different super rounds may contain different numbers of periods, as the membership list is continuously changing (we discuss node joins and leaves in Sec. IV-C).

Fig. 3 shows an example of Algorithm 1 in action. There are four active members in the network, aligned topologically in a straight line. $\|M_iM_r\| = \|M_rM_q\| = \frac{1}{2}\|M_qM_p\| = d$ (as Fig. 3a). Thus, M_r, M_q, M_p are $d, 2d$ and $4d$ away from M_i respectively. When $m = 1$, $\mathcal{P}_{M_i} = [\frac{1}{d}, \frac{1}{2d}, \frac{1}{4d}]$, $p_{min} = \frac{1}{4d}$. Thus, $\mathcal{BAG}_{M_i} = [4, 2, 1]$ respectively for M_r, M_q and M_p.

At the start of this super round, there are $1 + 2 + 4 = 7$ instances in the bag at M_i. Fig. 3b shows that for our protocol M_i sequentially pings M_r, M_p, M_q in the first three time periods. Then, M_p's instance is no longer in this bag (only 3 for M_r, and 1 for M_q left), so in Pass 2 M_i pings M_q and M_r in the next two time periods. Only 2 M_r instances are left. Passes 3 and 4 each pick one M_r for one period each. This concludes the super round for M_i, and new bag is created again for the next super round based on the latest $\mathcal{ML}_i$.



(a) Neighbor distances and initial full bag of M_i



(b) Target selection example of M_i

Fig. 3: **An example of time-bounded target selection in one super round (seven time periods in total, four members in network, $m = 1$).**

B. Time Bound

The approach above preserves relative ping selection probabilities because ping $Count_i$'s are normalized derivations of ping probabilities p_i. At the same time, this protocol provides time-bounded completeness, as we prove now.

Theorem 3. *In a system of N IoT nodes, consider a non-faulty node M_i and a faulty node M_j in M_i's membership list. Let α be the highest $Count_k$ in M_i's bag counts (i.e., $\mathcal{BAG}_{M_i}$). Then: Medley guarantees M_i detects M_j's failure within a number of pinging periods that is upper-bounded by:*

$$((N-2)\cdot\alpha) + (N-1)$$

Proof. The worst case occurs when: a) M_j has the lowest count (=1) in M_i's membership list (bag), i.e., M_j is the farthest node from M_i, b) all other $(N-2)$ nodes in M_i's bag share the same $Count_k$ value of α; and in the execution run: c) M_i creates a new bag, and the *first* node it pings is M_j, and d) this first ping succeeds but M_j fails right after.

From this point onwards: *(i)* M_i will spend the rest of this super round by executing $(N-2)\cdot\alpha$ periods pinging nodes other than M_j. At the start of the next super round, when M_i creates a new bag, the first pass will pick every node once, including M_j. Thus, the worst case occurs when M_j is picked *last* at the end of this first pass (in this next bag). This means: *(ii)* M_i will take another $(N-2)$ protocol periods to get around to pinging M_j. Finally: *(iii)* one additional protocol period is needed where M_i actually pings M_j.

Adding *(i), (ii), and (iii)*, the worst-case detection time of faulty node M_j at M_i is (in protocol periods):

$$((N-2)\cdot\alpha) + (N-1)$$

□

C. Node Joins and Removals

If a new node M_j is added to, or removed from, M_i's membership list just as the bag is about to be refilled, then all the members' ping probabilities (and thus $Counts$) are recalculated and normalized to reflect the changed membership. Additionally, Medley also allows node joins and removals in the midst of passes–the only rule required for correctness (to preserve relative ping probabilities) is to normalize the ping probability (and thus Counts) of the added/removed nodes to match current super round progress, based on the leftover nodes in the bag. When the bag becomes empty next, probabilities (and thus Counts) of all other members are recalculated and re-normalized anew.

V. Medley-F: Feedback-Based Target Selection

Medley, as described so far, may have a long tail of detection time for a small subset of nodes. We define an *unlucky node* n_i as one whose neighbors have all their (respective) neighbors much closer to themselves, while n_i is relatively far from each of its neighbors. When the exponent m is high and pings stay local, unlucky nodes have fewer pingers. If an unlucky node fails, its detection time will be longer than other nodes. To reduce this tail, we explore a variant of Medley, called Medley-F. Medley-F consists of competing approaches: an *active approach* wherein a node actively realizes it is unlucky, and a *passive approach* wherein other nodes realize the unlucky node. In both cases, the modified Medley adjusts the rate of pinging to the unlucky node—permanently for the active approach and temporarily for the passive approach.

A. Active Feedback Strategy

In active-feedback, every node *actively* monitors itself and reports its unluckiness to its 1-hop neighbors. These neighbors adjust their pinging probability to the unlucky node.

a) Member Self Monitoring: Each node estimates, via exponential averaging, the average interval of *incoming pings*. Given a new measurement M of pinging interval, Medley-F updates the estimate pinging interval I via exponential averaging: $I \leftarrow (1-\alpha)\cdot I + \alpha \cdot M$. We use $\alpha = 0.125$ in our implementation. If the estimate I is above `ACTIVE_TIMEOUT`, M_i considers itself as `UNLUCKY` and reports to all its direct neighbors. We recommend setting `ACTIVE_TIMEOUT` to be less than suspicion timeout (e.g. half of suspicion timeout), so that an unlucky node can report its unluckiness and potentially update its aliveness to other nodes in a timely manner.

b) Unlucky handling: When a node M_j receives an `UNLUCKY` report from a 1-hop neighbor M_i, M_j uses Algo. 2 to boost the pinging probability of M_i to reach the average pinging rate for other non-unlucky (or lucky) nodes.

We describe Algo. 2 via an example. Consider a node M_5 is maintaining pinging probabilities of [0.43, 0.3, 0.17, 0.1] for its four neighbors M_1 - M_4. Say M_5 receives an `UNLUCKY` report from M_4 ($p = 0.1$). Following Lines 1 - 3, the target (average) pinging probability from M_5 to M_4 will be $(0.43 + 0.3 + 0.17 + 0.1)/4 = 0.25$. Because M_1 and M_2 have above average probabilities, they become *probability sponsors* (Line 5). The excess probability of 0.18 + 0.05 = 0.23 is provided to boost M_4 from 0.1 to 0.25. Since 0.23 $>$ 0.15, M_1 and M_2 respectively and proportionally provide $0.18 \times \frac{0.15}{0.23} = 0.12$ and $0.05 \times \frac{0.15}{0.23} = 0.03$. The final probability list at M_5 is $[0.31, 0.27, 0.17, 0.25]$.

Algorithm 2 Probability modification for a single node M_j after receiving `UNLUCKY` reports of M_i

Require: M_j's Runtime Probability List $\mathcal{P}_{M_j}$

1: ▷ Get target probability for unlucky node M_i
2: $\mathcal{P}_{above}$ = all p_k in $\mathcal{P}_{M_j}$ with $p_k > p_{M_i}$ or $k = M_i$
3: p_{target} = mean($\mathcal{P}_{above}$)

4: ▷ Migrate probability from high-prob nodes to M_i
5: $\mathcal{P}_{sponsor}$ = all (p_k - p_{target}) in $\mathcal{P}_{M_j}$ with $p_k > p_{target}$
6: **for** each p_k in $\mathcal{P}_{sponsor}$ **do**
7: p_k -= (p_k - p_{target}) $\times \frac{p_{target} - p_{M_i}}{\text{sum}(\mathcal{P}_{sponsor})}$
8: **end for**
9: p_{M_i} = p_{target}

This approach boosts unlucky nodes and reduces pinging only to nodes with already-high ping rates. This approach also works with Sec. IV's bag strategy–instances in the current bag are updated immediately based on new probabilities.

B. Passive Feedback Strategy

The passive-feedback strategy is the reverse of active-feedback, and uses neighbors to detect an unlucky node. In passive-feedback, each node M_i actively maintains timestamp information about the *last contact* from other members. Such contact may be either through a direct contact where 1-hop neighbor sends or forwards a message, or via an indirect contact where a multi-hop member originates a message. To reduce the message payload, we do not keep information for intermediate routing path nodes. When selecting the next ping target, M_i flips a coin with probability $p_{passive}$, and if it turns up heads, M_i does a *passive check*. During a passive check, M_i looks at its membership list and checks whether any node has not contacted M_i in the last `PASSIVE_TIMEOUT` time units. If any, M_i suspects it as unlucky and randomly selects one of them as the next ping target. In our implementation we set to a less aggressive $p_{passive} = 0.1$.

Under the bag strategy of Sec. IV, the above selection does not remove any instance from the bag. We recommend setting `PASSIVE_TIMEOUT` larger than $t_{period} \times N$, where t_{period} is the pinging interval and N is network size, so that the ping target selection does not regress to uniform random pinging.

In active-feedback, the unlucky members that a node gets notified about are always 1-hop neighbors, while in passive-feedback the reported unlucky nodes are often "far" members whose information tends to stay local (at high m). While pinging such far nodes involves more hop-to-hop communication, passive-feedback can: i) still save considerable bandwidth compared to basic SWIM since the majority of ping targets are still local, and ii) avoid extra messages to report unlucky nodes that active-feedback needs.

VI. System Design

We now discuss practical considerations that were needed in order to implement Medley in a real IoT network.

Distance Metric: The analysis in Sec. III-B is based on physical distances. However, exact physical locations are hard to calculate; furthermore, physical distance may not be proportional to end to end (multi-hop) routing latency. As a result, our Medley implementation replaces the use of physical distance in the ping-probability equations (Sec. IV) with the metric of *hop-distance*. The hop-distance is the actual total distance that a message travels between two nodes, i.e., sum of distances of all intermediate hops.

For instance, if the locations of nodes M_1, M_2 and M_3 form an isosceles right triangle with $\|M_1M_2\| = \|M_2M_3\| = 1$. Suppose M_1 pings M_3 through M_2: Medley considers the distance between M_1 and M_3 as $\|M_1M_2\| + \|M_2M_3\| = 2$ instead of $\sqrt{2}$ which would have been the physical distance.

In our deployment experiments (Sec. VII), for comparison, we also implemented two alternative distance metrics. These are: 1) *latency metric*: actual end-to-end latency, and 2) *hop-number metric*: count of number of hops. We found that: a) the performance of Medley with latency metric was comparable to using the hop-distance metric, and b) Medley with hop-number metric behaves similar to hop-distance metric under grid topology. Thus hereafter we only show results using the hop-distance metric, with a few differing results shown using the hop-number metric.

***Other Medley Features*:** We clarify a few other features of Medley. First, the spatial probabilities we just described are for selecting not only ping targets, but also indirect pings. (Sec. II). Second, the rejoin of a failed node is considered as a new node. We denote the ID of each node with its IP address and local timestamp when it joins the network. Two IDs with the same IP but different join timestamps are considered as two incarnations. If M_i receives an active update for M_j with ID (ip_j, ts_1) that is different from its local record for M_j :

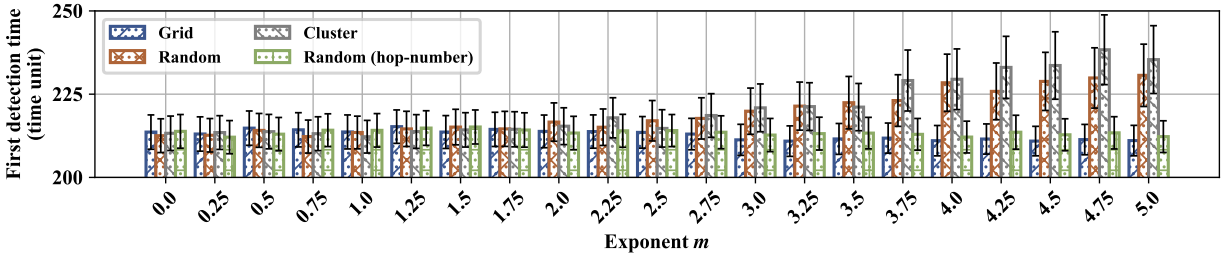


Fig. 4: **First failure detection time under different m and for 3 topologies. All use hop-distance metric, except Random (hop-number) which uses the hop-number metric.**

(ip_j, ts), M_i will consider the old incarnation as failed and continue with the latest ID for M_j. In practice this scenario occurs rarely as Medley dissemination times are fast.

VII. Experiments

We perform both simulations and deployments using Raspberry Pi devices. We present simulation results first in Sec. VII-A, and then deployment results in Sec. VII-B.

A. Simulation Results

The theoretical analysis of Sec. III made simplifying assumptions about uniformity and used physical distances. In this section, we explore realistic node layouts and measure the behavior and performance of our real Medley system.

There is a dearth of reliable simulators for IoT networks. We wrote our *first* simulation using NS-3 (v3.27), to be able to capture link layer effects [21]. However, NS-3 code cannot be deployed directly on Raspberry Pis. Thus for this current paper, we developed a *second matching* simulator, in Java, that *uses the same code as our Raspberry Pi deployment* but without NS3's fine-grained link layer modeling. We verified that the Java simulator's results match with our deployment at small scales, allowing us to use the simulator to extrapolate deployment results. Further, as the two simulators produced similar results, below we present only the Java simulation results (unless otherwise mentioned).

We evaluated Medley and Medley-F in three topologies: Random, Grid, and Cluster (multiple clusters in the area), each with 49 nodes deployed in $15m \times 15m$ area. The communication range for each node is $4m$. The default number of members chosen as indirect pinger was $K = 3$, and protocol period was 20 time unit. The suspicion timeout, `ACTIVE_TIMEOUT` and `PASSIVE_TIMEOUT` were set as 160, 80, and 400 time units in the experiments respectively. Each data point reflects data from 1000 independent runs. In every period, the probability to apply passive-feedback is 10%. Unless otherwise specified, Medley uses the hop-distance metric from Sec. VI.

1) Failure Detection and Dissemination Latency: We define *first detection time* as the time gap between a failure occurring and the first non-faulty node detecting this failure (after suspicion timeout). Fig. 4 shows how exponent m affects first detection time (averaged across 1000 runs), and square root of standard deviation. Across the three topologies using the hop-distance metric, Grid has the lowest detection time, with Random next, and Cluster the worst. In Random and Cluster topologies, there might be unlucky nodes (Sec. V). When m is high, pings stay local, and unlucky nodes are pinged less frequently, thus prolonging their detection times. Comparably, Grid is more deterministic in assigning every node at least a small set of neighbors at short distances, producing more stable detection times. This result is different from Sec. III-B's analysis because: i) the node layout and density assumptions are different, and ii) use of hop-distance metric (Sec. VI) to calculate ping probabilities.

For Cluster and Random topologies, *first detection time* stays low for $m \leq 3.5$ and rises quicker when $m \geq 3.5$. This is due to that: i) a quick increase (with rising m) in initial bag size increases the duration of a super round (Sec. IV), and ii) unlucky node's lower probability (fewer instances in bag) to be as a ping target by any of its neighbours. When m is low enough (below 3.5), the bag sizes are manageable and nodes have sufficient pingers for fast failure detection. Beyond $m = 3.5$, the bag size increases quickly, and thus super round length. It takes much longer for a pinger to pick a failed unlucky node, which could be as long as a super-round in the worst case (Theorem 3). We also observe from Fig. 4 that Medley using the hop-number metric in the Random topology behaves similar to Grid topology, with relatively stable first detection time. The reason is that each member has at least one one-hop (shortest distance) neighbour as a pinger, making unlucky nodes rarer than when using hop-distance metric.

Dissmination Latency and Active & Passive-feedback Optimizations: We measure *dissemination time*, the time for all nodes to know about a failure after the first detection. Fig. 5 stacks dissemination time atop detection time. For a fair comparison, instead of raw first detection time, in (only) this plot we use *differential first detection time (dfdt)* = (*first detection time*) minus (*minimum detection time*). The (theoretical) *minimum detection time* in our deployment is 180 time units: a sum of detection timeout of 20 time units, and suspicion timeout of 160 time units.

We find that active-feedback (only) is the most effective at reducing *dfdt* by up to 31.1%. Active and passive together reduce by 27.4% and passive-feedback-only by 11.5%. Combining both active and passive provides 54.6% reduction in dissemination time (*dsm*), with passive-only at 44.5% and active-only at 31.6%. Intuitively, active-feedback-only offers the shortest and stablest first detection time even under high m, since it helps each *unlucky* node get frequent pings. For dissemination, intuitively, the combination of active-feedback

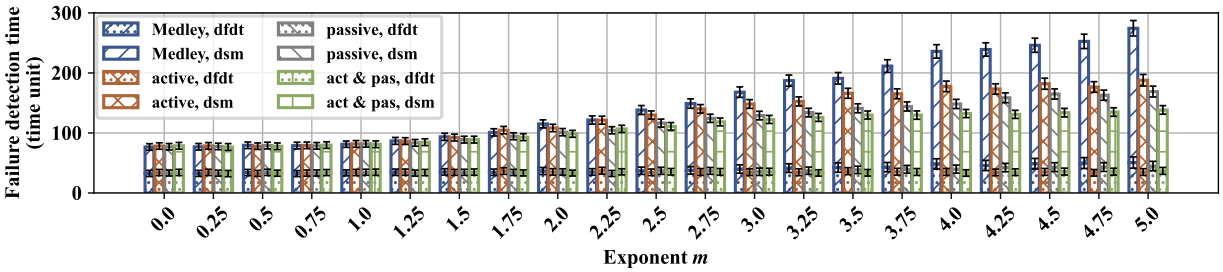


Fig. 5: **Differential first detection time (dfdt), stacked dissemination time (dsm). Various m and optimizations. Random topology. Hop-distance metric.**

spreading locally and passive-feedback spreading far away, is fastest. Overall, at m=3, we recommend combining active and passive, to reduce P95 *dfdt* by 31.2% and *dsm* by 47.2%.

Simultaneous Failures: We simultaneously fail 50% (randomly chosen) nodes (24 out of 49) in the Random topology. Fig. 6 shows the average first detection time. The lower and higher error bars are respectively the *earliest* and *latest* time any failure is detected, averaged across runs.

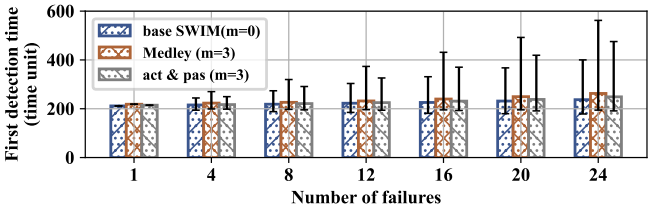


Fig. 6: **Failure detection time under simultaneous failures (Random topology)**

The average (raw) first detection time of Medley and its variants rises gently as m and number of failures increase. As m rises, a failed unlucky node waits longer to become a ping target because pings stay local. Now, define the *detection gap* as the percentage by which detection time is prolonged under massive failure (50% nodes) vs. just a single failure scenario. In base SWIM (at $m = 0$), the detection gap is only 12.3%– due to the uniform randomness, a failed node has a high probability $1 - (\frac{48}{49})^{24} \simeq 39.0\%$ of being pinged each round after failure. In Medley, the detection gap is 20.6% due to localized pings and unlucky nodes' higher detection times. Applying active and passive feedback reduces the gap to 15.8%. Note that Medley's slightly longer detections come with massive bandwidth savings (Sec. VII-A2).

Domain Failure: Next, we explore the effect of massive failures in an area (e.g., connected to a power breaker). 49 nodes are located in five clusters in the square area of interest. Each run randomly fails a whole cluster.

From Fig. 7 (bars similar to Fig. 6) we observe that the average first detection time stays low when $m < 2$, and as expected it increases as m rises. The increase in detection time with m is because of the ping localization under higher m, implying that the typical way a detection proceeds at higher m is from the edges of the failed cluster towards the cluster's middle. In comparison, lower values of m would detect nodes near the middle of the failed cluster much quicker due to the higher probability of far-away non-faulty pingers. Under high

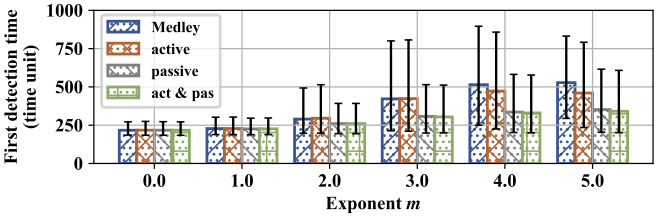


Fig. 7: **Failure detection time under domain failures (Cluster topology).**

m, both active and passive reduce detection time, with passive more effective since it provides a higher chance to detect nodes near the "middle" of the failed cluster.

2) Communication Cost: Fig. 8 shows the CDF of the hop count of messages. Point(x, y) means $y\%$ of messages travel fewer than x hops. As expected, lower m (basic SWIM with uniform pinging) incurs far more hops, while Medley localizes traffic. Active feedback does not affect traffic much, since the ping probability modification occurs only among nodes with already-high pinging probabilities, i.e., already close to pinger. Passive feedback raises traffic as farther nodes are affected.

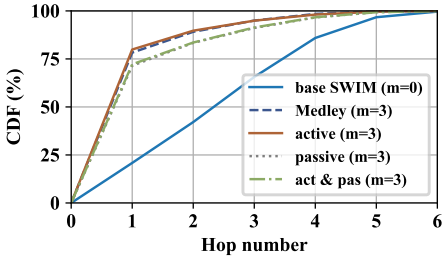


Fig. 8: **CDF of hop numbers that messages travel under different strategies in Random topology.**

Since Medley's goal is to minimize both communication cost (messages sent, counting multiple hops) and detection time, we measure the square root of their product in Fig. 9, for Random topology. Medley's product cost (lower is better) falls quickly as m goes from 0 to 3, and then slowly rises. At lower m, Medley pings faraway nodes more frequently. Rising m increases detection time (Fig. 4), yet the associated communication drop (due to localization) is faster. At higher m, communication cost reduction slows down, and thus detection time increase dominates. Compared to base SWIM ($m = 0$), Medley's product cost is 35.2% lower.

With active-feedback, Medley-F's product cost (under m=3) is 37.8% lower than base SWIM, as active-feedback lowers

communication cost effectively. Applying passive-feedback, and active+passive, do not bring higher benefits, since passive sends faraway pings. However, product cost is still lowered by up to 30% and 31.2% respectively compared to basic SWIM. At $m > 3$, in all feedback-based strategies, communication reduction balances out detection time increase.

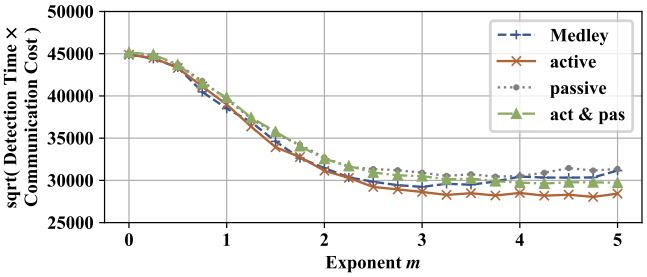


Fig. 9: **Average failure detection time × Communication cost, Square Root (Random topology). Lower is better.**

3) False Positive Rate: We measure the rate of false detections, which are non-faulty nodes mistakenly detected as failed (this may occur due to slow nodes, dropped packets, etc.). Because false detections are affected by link layer behaviors, we use the high-fidelity NS3 simulator under 25 nodes. In Table II, we drop a random fraction r_{loss} of packets (on hops). We measure false positive rate as the fraction of time, over the entire run, that a false positive detection persists, i.e., fraction of time that at least one non-faulty node is considered failed by at least one other non-faulty node.

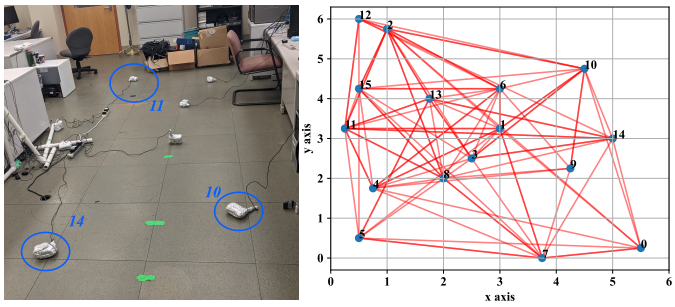
In Table II, higher packet loss rates imply higher false positive rates, as expected. We also observe that false positive rate drops with increasing m (for a given packet loss rate). This is because at lower m, pings and acks have to transit more hops, thus increasing the chances that at least one of the hops will drop the packet, and a non-faulty node will be detected as failed due to a timeout. Further, at higher m, the suspicion (Sec. III-A) arising from a failure detection has a higher chance of being resolved due to the more repetitive and localized nature of pings.

TABLE II: **False positive rate under different packet loss rates (r_{loss}) and exponents m.**

r_{loss} \ m	0	1	2	3	4	5
10%	1.07%	0.43%	0.69%	0.08%	0.08%	0.00%
20%	2.32%	2.35%	2.05%	1.49%	1.36%	1.39%

B. Deployment Evaluation

We implemented a prototype of Medley in the Raspberry Pi (RP) [22] environment. Our Java implementation was around 3000 lines of code, under Raspbian 4.19. We deployed Medley in a network of 16 IoT devices in our lab space. Figs. 10a and 10b show a photograph and a map of one of our topologies. This random topology was in a $6m \times 6m$ area (grid lines only for reference purposes). The routing protocol is OLSR [23]. Each device was a Raspberry Pi 4 model B, with 2GB LPDDR4 RAM and Broadcom BCM2711, 1.5 GHz quad-core Cortex-A72 CPU. Since the signal strength of Pis were too strong to make multi-hop routing with respect to the limited deployment area, we attenuated each Pi by both: a) consistently wrapping in aluminum foil, and b) setting transmit power to 15 dBm, to force more multi-hop transmissions. Red lines Fig. 10b is a screenshot of routine paths. Prior to these experiments, we performed benchmark experiments to verify that this attenuation was stable and consistent across Pis.



(a) Lab placement (b) Deployment topology (Random)

Fig. 10: **Topology of Raspberry Pi deployment.**

1) Failure Detection and Dissemination Latency: From Fig. 11 we observe that failure detection time and dissemination time both increase as m becomes larger. (The plot used 32 data points per failure, with average and standard deviation shown.) This is because disseminating failure information of unlucky nodes (e.g., nodes 0, 9 in Fig. 10) takes a while since spatial pinging (hence piggybacking of failure information) stays largely local especially at high m. Similar to simulation results, both active-feedback and passive-feedback produce benefits for first detection time and dissemination time. From the simulation (Sec. VII-A), we expected active-feedback to work the best for first detection time and passive-feedback to be effective on dissemination latency reduction. In the deployment, active-feedback and applying both strategies do act as expected. However, at high m, passive-feedback performs poorly on dissemination time, because benefits of multi-hop dissemination do not emerge in our smaller deployment scales. passive-feedback may only be more preferable at larger scales.

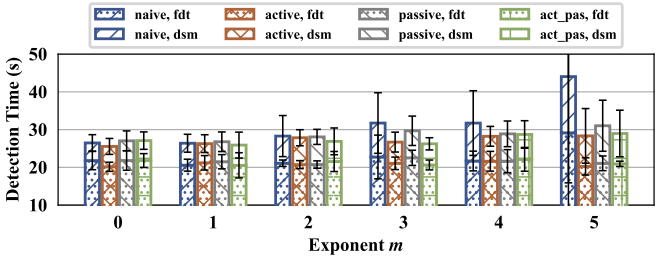
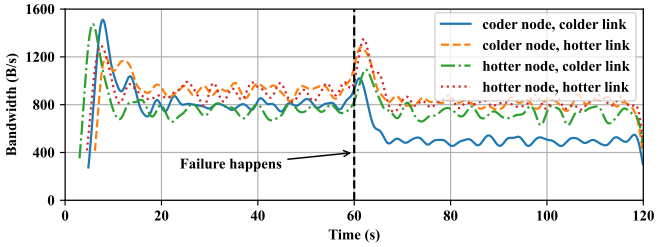
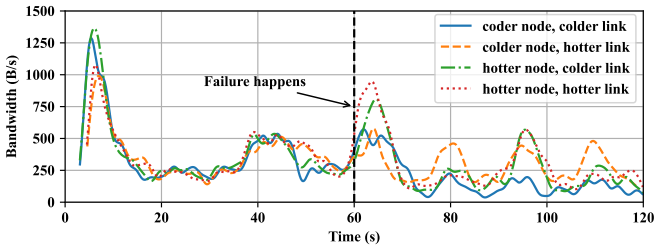


Fig. 11: **First failure detection time and dissemination time for Raspberry Pi experiments.**

2) Bandwidth Cost over Time: We denote links that lie on more routing paths (of node pairs) as *hotter links*, and those on fewer paths as *colder links*. For simplicity we used a smaller topology and a fixed routing table with 7 Raspberry Pi [21]. Fig. 12 plots real-time bandwidth on a hotter link and a colder link. In each run a node fails at time 60 (a hotter node or a colder node). Compared to $m = 0$, $m = 3$ consumes lower bandwidth on average (61.8% less for hotter link, and 52.9% less for colder link), but fluctuates inside a super round.



(a) **Base SWIM ($m = 0$). Hop-number metric.**



(b) **Medley ($m = 3$). Hop-number metric.**

Fig. 12: **Bandwidth change timeline. Failures occur at $t = 60$. Using the hop-number metric.**

Both far pings and local pings tend to go through hotter links. Bandwidth cost is high right after new nodes join (time 5 to 10) and right after failures occur (time 60 to 70)—this is due to increase in indirect pings. Larger exponent values (m) mean that a failure will cause bandwidth to rise more ($3\times$ at $m = 3$ and $1.5\times$ at $m = 0$). Yet the peak bandwidth consumption in Medley ($m = 3$) stays lower than base SWIM's ($m = 0$).

At high m bandwidth usage has a periodic behavior caused by the cyclical nature of the super-round. Fig. 13 depicts the bandwidth and FFT for a 700 s run with no failures. We first observe that the bag selection strategy does not affect *average* bandwidth. Second, the random selection from bag has lower bandwidth fluctuation over time, while pass-based has bigger amplitudes. This is because in the pass-based approach (Algorithm 1), the pings in the second half of each super-round tend to focus on close neighbors (a small group of nodes which have higher counts in the bag), leading to temporally unbalanced communication load on links. In comparison, selecting from the bag targets at random (rather than via passes) has less pronounced periodicity.

Although random strategy benefits from balanced bandwidth, it has longer detection times: $2\times((N-2)\cdot\alpha)+1$ periods, almost twice as pass-based (for high m). If the application prefers reducing detection time than minimizing bandwidth, the pass-based approach is preferable.

VIII. Discussion

***Partial Membership Lists*:** Medley maintains full membership lists, useful for building a swath of distributed algorithms (Sec. I). Nevertheless, full membership lists can be "pared" down to partial membership lists, without affecting properties, while reducing overhead. Two examples follow. Ex. 1: If a multicast tree (built atop Medley) uses only nearby neighbors, the partial membership list can maintain mostly nearby neighbors. Ex. 2: It is well-known that uniformly-randomly-selected partial membership lists give identical properties as a full list, for gossip multicast applications [24]. For this case,

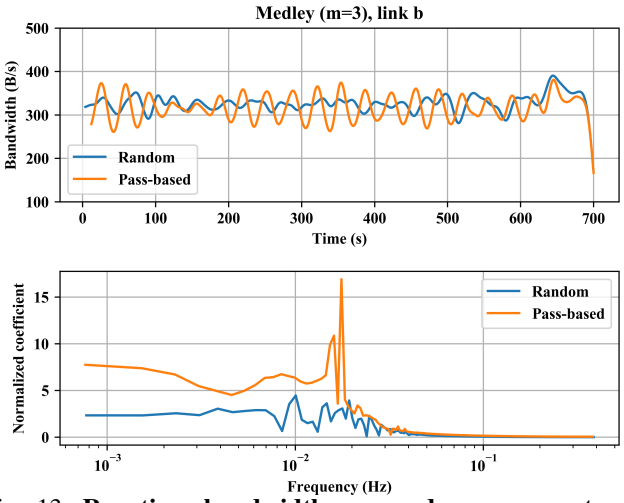


Fig. 13: **Run-time bandwidth on random vs pass target selection under Medley ($m = 3$). Top: Bandwidth. Bottom: Bandwidth's FFT.**

Medley's partial membership lists could be built in one of two ways: i) apply a uniform-random selection strategy to pick the partial membership list, and use spatial pinging, or ii) apply the spatial distribution to pick the partial membership list, and use uniform-random pinging. In both cases [24] would extend, meaning that gossip over Medley with partial lists would behave identically as gossip over Medley with full lists.

***Topology Optimizations*:** An open direction is leveraging knowledge of network topology. For instance, one could avoid intersecting routes for pings, route pings/acks avoiding failure domains, and avoid routine via failed nodes.

IX. Related Work

Classical Failure Detection: Failure detection in data-centers is well-studied. The earliest failure detectors send periodic "I am alive" heartbeats [11] to all other or a subset of nodes. Timeout on the next heartbeat leads to failure detection. Heartbeats may be multicast or gossiped [12] or spread hierarchically [25]. As described earlier, SWIM [9] is the inverse of heartbeating, relying on pinging, and has bandwidth provably within a constant factor of optimal. FUSE [15] disseminates failure information via applications, to reduce network costs.

Failure Detection in IoT Networks: Existing IoT failure detection schemes largely focus on data anomalies and can be used orthogonally with Medley. Sympathy [26] uses flooding and aggregates distributed data at the sink, detecting failure by finding insufficient flow of incoming data. Memento [27] uses a tree for failure monitoring, limiting its scalability under failures. Network-level delays and packet traces can be used for failure detection [28], [29]. Yet, these are hard to analyze mathematically. DICE [6] uses context (e.g., sensor correlation, state transition probabilities) to identify anomalous readings and their sensor nodes. All the above works can be used orthogonally with Medley. Asim et al. [30] partitions the network into cells, detects failures within cells, and multicasts it across cells–this however assumes a homogeneous network.

X. Conclusion

We have presented design, analysis, and implementation of Medley, a decentralized membership service for distributed

IoT systems running atop wireless ad-hoc networks. Our key idea is a spatial failure detector, that prefers pinging nearby nodes with an exponentially higher probability. Compared to classical SWIM, Medley and its variants detects failures just as quickly, while lowering the product of failure detection time and communication cost by 37.8%, and incurring low false positive rates around 2% even with 20% dropped packets. Active and passive feedback reduce tail detection time by up to 31%, and dissemination time by up to 54%.

Code is available at: http://dprg.cs.uiuc.edu/downloads.php

Acknowledgments. This work was supported in part by each of the following: NSF CNS 1908888, a Capital One gift, and a Microsoft gift.

REFERENCES

[1] MarketsandMarket™, "Internet of Things (IoT) market by software solution (real-time streaming analytics, security solution, data management, remote monitoring, and network bandwidth management), service, platform, application area, and region - global forecast to 2022," MarketsandMarkets™, 2018. [Online]. Available: https://www.marketsandmarkets.com/PressReleases/iot-m2m.asp

[2] D. Watkins, "Global smart speaker shipments and revenue by model and price band: Q2 2018," Strategy Analytics, 2018. [Online]. Available: https://tinyurl.com/smart-speaker-revenue

[3] K. Kapitanova, E. Hoque, J. A. Stankovic, K. Whitehouse, and S. H. Son, "Being smart about failures: assessing repairs in smart homes," in *Proc. of ACM UbiComp*, 2012, pp. 51–60.

[4] J. Ye, G. Stevenson, and S. Dobson, "Detecting abnormal events on binary sensors in smart home environments," *Pervasive and Mobile Computing*, vol. 33, pp. 32–49, 2016.

[5] A. K. Sikder, H. Aksu, and A. S. Uluagac, "6thsense: A context-aware sensor-based attack detector for smart devices," in *Proc. of USENIX Security*, 2017, pp. 397–414.

[6] J. Choi, H. Jeoung, J. Kim, Y. Ko, W. Jung, H. Kim, and J. Kim, "Detecting and identifying faulty IoT devices in smart home with context extraction," in *Proc. of IEEE DSN*, 2018, pp. 610–621.

[7] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46," *Official Journal of the European Union (OJ)*, vol. 59, no. 1-88, p. 294, 2016.

[8] "Health insurance portability and accountability act of 1996," *Public Law*, vol. 104, p. 191, 1996.

[9] A. Das, I. Gupta, and A. Motivala, "SWIM: Scalable weakly-consistent infection-style process group membership protocol," in *Proc. of IEEE DSN*, 2002, pp. 303–312.

[10] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *Proc. of ACM SOSP*, 1987, pp. 123–138.

[11] M. K. Aguilera, W. Chen, and S. Toueg, "Heartbeat: A timeout-free failure detector for quiescent reliable communication," in *WDAG*. Springer LNCS 1320, 1997, pp. 126–140.

[12] R. Van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Proc. of Middleware*, 2009, pp. 55–70.

[13] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 system architecture," *IBM Systems Journal*, vol. 34, no. 2, pp. 414–446, 1995.

[14] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, "On scalable and efficient distributed failure detectors," in *Proc. of ACM PODC*, 2001, pp. 170–179.

[15] J. Dunagan, N. J. Harvey, M. B. Jones, D. Kostic, M. Theimer, and A. Wolman, "FUSE: Lightweight guaranteed distributed failure notification," in *Proc. of USENIX OSDI*, 2004.

[16] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona, "The real Byzantine generals," in *Proc. of IEEE DASC*, vol. 2, 2004, pp. 6–D.

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

[18] L. Lozinski, "How Ringpop from Uber Engineering helps distribute your application," 2016. [Online]. Available: https://eng.uber.com/intro-to-ringpop/

[19] Serf, "Serf by HashiCorp: Decentralized cluster membership, failure detection, and orchestration," 2014. [Online]. Available: https://www.serf.io/

[20] Consul, "Consul by HashiCorp: A distributed service mesh to connect, secure, and configure services across any runtime platform and public or private cloud," 2014. [Online]. Available: https://www.consul.io/

[21] R. Yang, S. Zhu, Y. Li, and I. Gupta, "Medley: A novel distributed failure detector for iot networks," in *Proc. of Middleware*, 2019, pp. 319–331.

[22] "Raspberry Pi 4 model B," 2016. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[23] "Optimized link state routing protocol." [Online]. Available: https://tinyurl.com/olsrd-wiki

[24] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 139–149, 2003.

[25] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh, "Efficient epidemic-style protocols for reliable and scalable multicast," in *Proc. of IEEE SRDS*, 2002, pp. 180–189.

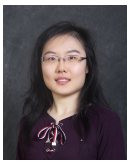
[26] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," in *Proc. of ACM SenSys*, 2005, pp. 255–267.

[27] S. Rost and H. Balakrishnan, "Memento: A health monitoring system for wireless sensor networks," in *Proc. of IEEE SECON*, vol. 2, 2006, pp. 575–584.

[28] B. Chen, G. Peterson, G. Mainland, and M. Welsh, "Livenet: Using passive monitoring to reconstruct sensor network dynamics," in *Proc. of DCOSS*. Springer LNCS 5067, 2008, pp. 79–98.

[29] R. N. Duche and N. P. Sarwade, "Sensor node failure detection based on round trip delay and paths in WSNs," *IEEE Sensors Journal*, vol. 14, no. 2, pp. 455–464, 2014.

[30] M. Asim, H. Mokhtar, and M. Merabti, "A fault management architecture for wireless sensor network," in *Proc. of IEEE IWCMC*, 2008, pp. 779–785.



Rui Yang is a Ph.D. candidate in the department of Computer Science at the University of Illinois at Urbana-Champaign, working with Prof. Indranil Gupta. She is interested in reliability and resource scheduling for distributed system in general. Her recent work studied the responsiveness-correctness-cost tradeoffs for both system-facing and user-facing substrates in the Internet-of-Things (IoT) scenarios.

Jiangran Wang graduated in 2021 from University of Illinois at Urbana-Champaign with a Bachelor's degree in Computer Engineering. He is pursuing a M.S. degree in Electrical and Computer Engineering at Urbana-Champaign. His research focus on distributed systems, primarily on the Internet of Things and failure detection algorithms.

Jiyu Hu graduated in 2021 from University of Illinois at Urbana-Champaign with a Bachelor's degree in Computer Engineering. His research focus was Distributed Systems, primarily on IoT and Edge Computing. He now pursues a Master of Computational Data Science degree from the School of Computer Science at Carnegie Mellon University. His area of interest remains in data-intensive systems.

Shichu Zhu completed his M.S. degrees in Computer Science and Atmospheric Science at the University of Illinois at Urbana-Champaign in 2019. He has research experience in distributed systems, databases and HCI, as well as the microphysics of Cirrus clouds. He is currently working in Google Cambridge on Cloud Networking.

Yifei Li graduated in 2019 with a Master of Science in Computer Engineering degree at the University of Illinois, Urbana-Champaign. His prior research focus in distributed systems was around databases and the Internet of Things. He currently works for Confluent Inc, which is a leading force in event streaming platforms. He primarily works on Inter-cluster data replication and cloud services.

Indranil Gupta is a Professor of Computer Science at UIUC, an IEEE Senior Member and an ACM Distinguished Scientist. He leads the DPRG research group (http://dprg.cs.uiuc.edu/) that works on large-scale distributed systems. He has also worked at Google, IBM Research, and Microsoft Research. His PhD is from Cornell (2004) and B. Tech from IIT Chennai (1998).