Medley: A Membership Service for IoT Networks
Rui Yang®, Jiangran Wang®, Jiyu Hu®, Shichu Zhu, Yifei Li, and Indranil Gupta®, Senior Member, IEEE

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 other critical infrastructures such as smart cities, smart hospitals, smart forests, battlefield scenarios, etc., are proliferating. 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). 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., 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 does not use excessive memory. For instance, even with up to 5K devices

1Malicious/Byzantine failures are outside our scope.

Digital Object Identifier 10.1109/TNSM.2022.3196268
This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
(sufficient for a multi-storey building), if each membership entry is 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 with 2 GB of memory, the membership list would occupy only 0.005% of node 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] 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 [9], [14] 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 (SWIM), Medley provides comparable failure detection times, lowers bandwidth by 37.8% (given a detection time), and has false positive rates of 2% under 20% packet drop rates. We cut tail detection time 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 synchrony, 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.

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 sends each of them a ping-req($M_j$) message which requests 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 all membership information. Analysis [14] shows that in a system with $N$ nodes, information spreads with high probability to all nodes within $O(\log(N))$ time periods.

Using a Membership List in IoT: Full membership has long been used as a building block in Internet-wide distributed systems for building reliable and fast protocols [12], from leader election and multicast trees to consensus. Medley enables the same class of protocols to be designed over IoT networks, without the need to rely on any kind of centralization. These protocols are already used in IoT networks for spreading commands and for coordination. Medley allows these protocols to work in scenarios without access to clouds (e.g., battlefield, environmental observation systems), or where low latency operations are needed (e.g., smart farms), or where centralization is infeasible (e.g., with data privacy constraints such as hospitals). The fundamental tradeoff that Medley enables is to impose a low background bandwidth in maintaining these membership lists, so that these other protocols, on top of membership, can be run quickly and with minimal message exchange. Centralized and “cluster”-based approaches can incur delay in communication or on-demand cluster selection. For instance, a 1 RTT leader election protocol with $(N - 1)$ messages can be designed atop Medley’s full membership lists–each node selects the lowest ID (for instance) node in its membership list, and the leader multicasts an “I am the leader” message to all. We believe this is the right direction for IoT deployments—Medley begets simpler probabilistic protocols without corner cases, thus making it easier to deploy, debug, measure, and optimize IoT applications.

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 Section 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 $0.6$, $0.28$, 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 Section 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.

**Other Components:** Just like SWIM, Medley disseminates information by piggybacking atop pings, acks, and indirect pings (Section II, “SWIM Dissemination Component”). This is a gossip style of dissemination and is also used to disseminate node join/leave information. When a new node joins, it: i) borrows the membership list from any of its 1-hop neighbors, and ii) starts piggybacking its information atop the dissemination component, and thus becomes included in other nodes’ membership lists.

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 $E[\text{Pings received per period}]$ or $EP(m)$, where:

$$EP(m) = \left( \int_0^R \frac{1}{r^m} D(2\pi r) \cdot dr \right) \cdot \frac{1}{\pi R^2 D}$$

$$= \left( \int_0^R \frac{1}{r^{m-1}} \cdot dr \right) \cdot \frac{2}{R^2} \quad (1)$$

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:

$$C(m) = \left( \int_0^R \frac{1}{r^m} D(2\pi r) \cdot r \cdot dr \right)$$

$$= \left( \int_0^R \frac{1}{r^{m-2}} \cdot dr \right) \cdot (2\pi D) \quad (2)$$

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: $\text{Ratio}(m) = \frac{EP(m)}{C(m)}$.

**Theorem 1:** Medley’s spatial pinging: (i) provides completeness, and (ii) optimizes $\text{Ratio}(m)$ 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 subsequent 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.
TABLE I

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

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

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

where $d$ represents the distance to the nearest node (for a 2-dimensional deployment, $d \approx \frac{1}{\sqrt{D}}$).

Similarly, for communication cost $C(m)$:

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

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

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

1) $m = 0$ can be ignored as $\text{Ratio}(m = 1)$ is higher than $\text{Ratio}(m = 0)$.

2) $m = 2$ can be ignored as $\frac{x}{\log(x)}$ has a minimum of $e$ ($> 1$).

3) $m = 1$ can be ignored as $\frac{\text{Ratio}(3)}{\text{Ratio}(1)} = \frac{x}{2\log(x)}$ has a minimum at $\frac{e}{2} > 1$.

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

1) If $R \gg d$ (in particular $\frac{R}{d} > e \simeq 2.718$ or $\log\left(\frac{R}{d}\right) > 1$), 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 $\frac{R}{d} < e \simeq 2.718$, $m = 3$ 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, 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 $M_1$ 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 $M_1$ 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 $O\left(\frac{1}{e^{-1}}\right)$ 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 $O\left(\frac{1}{e^{-1}}\right)(N)$ 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.

A. Design of Time-Bounded Medley

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

$\text{Note that this is a different assumption from the analysis in Equation (3), but is closer to our real implementation.}$
Algorithm 1 Time-Bounded Target Selection in a Super Round From a Single Node $M_i$’s View

Require: Runtime Probability List $P_{M_i}$

1: $\triangleright$ Super round: Create initial bag $BAG_{M_i}$
2: for each $p_j$ in $P_{M_i}$ do
3: Put $[\frac{p_j}{p_{\text{min}}}]$ into $BAG_{M_i}$
4: end for
5: Create an empty set onePassTargets
6: $\triangleright$ Start target selection
7: while $BAG_{M_i}$ is not all zeros do
8: if onePassTargets is empty then
9: $\triangleright$ One Period
10: Randomly pick one node in onePassTargets as PING target
11: Remove $M_j$ from onePassTargets
12: Reduce $BAG_{M_i}(M_j)$ by one
13: end if
14: end while

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

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

Let $p_{\text{min}} = \min\{P_{M_i}\}$, the lowest probability among all non-faulty members in $ML_i$. Now, denote $Count_j = [\frac{p_j}{p_{\text{min}}}]$.

We create a initial bag list as:

$BAG_{M_i} = \{Count_1, Count_2, \ldots, Count_K\}$ (Line 1 - 3).

The weighted round-robin pinging at node $M_i$ creates a bag $B_j$ which consists of $Count_j$ instances of node $M_j$ for each $M_j \in ML_i$. 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 $\lceil \Sigma_{j=1}^K Count_j \rceil$ 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 Section 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: $\|M_1M_2\| = \|M_2M_3\| = \frac{1}{2}\|M_3M_4\| = d$ (as Fig. 3(a)). Thus, $M_r, M_q, M_p$ are $d, 2d$ and $4d$ away from $M_i$ respectively. When $m = 1$, $P_{M_i} = [\frac{1}{d}, \frac{1}{2d}, \frac{1}{4d}]$, $p_{\text{min}} = \frac{1}{4d}$. Thus, $BAG_{M_i} = [4, 2, 1]$ 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. 3(b) shows that for our protocol $M_i$ sequentially pings $M_r, M_q, M_p$ 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 $ML_i$.

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 $\alpha$ be the highest $Count_j$ in $M_i$’s bag counts (i.e., $BAG_{M_i}$). 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 \( \alpha \); 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 (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.

1) 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.

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

We describe Algorithm 2 via an example. Consider a node \( M_5 \) is maintaining pinging probabilities of \([0.43, 0.3, 0.17, 0.25]\) for its four neighbors \( M_1 - M_4 \). Say \( M_5 \) receives an UNLUCKY report from \( M_4 \) with probability \( \alpha = 0.1 \). 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 \( M_4 \) to 0.25. Since 0.23 > 0.15, \( M_1 \) and \( M_2 \) respectively and proportionally provide 0.18 \times 0.15 = 0.12 and 0.05 \times 0.15 = 0.03. The final probability list at \( M_5 \) is \([0.31, 0.27, 0.17, 0.25]\).

This approach boosts unlucky nodes and reduces pinging only to nodes with already-high ping rates. This approach also works with Section 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 \( P_{\text{passive}} \) 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

\[
\text{Algorithm 2 Probability Modification for a Single Node } M_j \text{ After Receiving UNLUCKY Reports of } M_i
\]

\begin{algorithm}
\begin{algorithmic}[1]
\Require \( M_j \)’s Runtime Probability List \( P_{M_j} \)
\State \( p_{\text{above}} = \) all \( p_k \) in \( P_{M_j} \) with \( p_k > p_{M_i} \) or \( k = M_i \)
\State \( p_{\text{target}} = \text{mean}(p_{\text{above}}) \)
\State Migrate probability from high-prob nodes to \( M_i \)
\State \( P_{\text{sponsor}} = \) all \( (p_k - p_{\text{target}}) \) in \( P_{M_j} \) with \( p_k > p_{\text{target}} \)
\For {\( p_k \) in \( P_{\text{sponsor}} \)}
\State \( p_k \leftarrow (p_k - p_{\text{target}}) \times \frac{p_{\text{target}} - p_{M_i}}{\text{sum}(P_{\text{sponsor}})} \)
\EndFor
\State \( p_{M_i} = p_{\text{target}} \)
\end{algorithmic}
\end{algorithm}

\[
\begin{align*}
\text{Require: } & \text{ } p_{M_i} \text{’s Runtime Probability List in } P_{M_i} \\
\text{End For } &
\end{align*}
\]
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 $p_{\text{passive}} = 0.1$.

Under the bag strategy of Section IV, the above selection does not remove any instance from the bag. We recommend setting $\text{PASSIVE\_TIMEOUT}$ larger than $t_{\text{period}} \times N$, where $t_{\text{period}}$ 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 Section 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 (Section 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. This can be measured during bootup via messages between all 1-hop node pairs, e.g., by [21].

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 $M_2$ as $\|M_1M_2\|+\|M_2M_3\| = 2$ instead of $\sqrt{2}$ which would have been the physical distance.

In our deployment experiments (Section VII), for comparison, we also implemented two alternative distance metrics: 1) latency metric: actual end-to-end latency (which can vary significantly over time, due to link characteristics), and 2) hop-number metric: count of number of hops. Over multiple experiments, we found that: a) Medley with latency metric was comparable to 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 hop-distance metric, with a few differing results shown using.

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. (Section 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 $(ip_j, ts_j)$ that is different from its local record for $M_j : (ip_j, ts)$, $M_i$ 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 Section VII-A, and then deployment results in Section VII-D.

A. Simulation Results

The theoretical analysis of Section 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 [22]. 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 both: 1) our deployment at small scales, allowing us to use the simulator to extrapolate deployment results; and 2) NS3 results at small scales. Hence we present only the Java simulation results.

We evaluated Medley and Medley-F in three topologies: i) Random (nodes are randomly placed), ii) Grid (7x7 grid), and iii) Cluster (there are 5 clusters with 7, 7, 9, 10, 16 nodes respectively where each cluster is bounded by a fixed square area), each with 49 nodes deployed in $15m \times 15m$ area. The communication radius for each node is $4m$. The random and cluster topologies are newly generated (new seed) for each trial run. The default number of members chosen as indirect pinger was $K = 3$, and protocol period was 20 time units. The suspicion timeout, $\text{ACTIVE\_TIMEOUT}$ and $\text{PASSIVE\_TIMEOUT}$ 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 Section 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, (after suspicion timeout), $\text{ACTIVE\_TIMEOUT}$ and $\text{PASSIVE\_TIMEOUT}$ 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 Section VI.

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 Section III-B’s analysis because: i) the node layout and
density assumptions are different, and ii) use of hop-distance metric (Section 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 (Section 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.

**Dissemination 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 stabllest first detection time even under high \( m \), since it helps each unlucky node get frequent pings. For dissemination, intuitively, the combination of active-feedback 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. 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 \( 1 - \left( \frac{48}{49} \right)^{24} \approx 39.0\% \) 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, which will be shown later in this section.

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 \( m \), 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.

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.

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. Each experimental run was identically long at 300K time units, so trends would remain unchanged if we replaced communication cost, i.e., messages, with bandwidth use, i.e., messages per second, or messages per second per hop. 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.

2) 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 (Section III-A) arising from a failure detection has
and reaches a steady state. Then we instantly move a fraction of the nodes in random directions by distances randomly uniformly chosen among $[0, \min(X \text{ meters, distance to edge of square})]$ ($X$ is the x axis value on the plots). With this move, we do not update Medley’s internal topology-related data at any nodes: distances, ping probabilities, etc., all remain stale from the pre-move topology. However, routing tables are updated post-move, as would be expected in a mobile network so that packets can be routed correctly.

In Fig. 11, the solid lines show the post-move performance of Medley, operating still with stale (pre-move) internal data structures. Dotted lines show Medley with corrected post-move internal data. Essentially, the gap between the solid and dotted lines shows the effect of Medley continuing with stale data structures. We observe that: a) at small mobility up to $X = 4$ meters, even with up to 40% of nodes moved, stale data does not affect Medley performance; b) when few nodes move (10% plot), larger mobility distances can be tolerated—the metric rises by at most 7.1% at $X = 16$; c) when more nodes move (40% plot), metric degradation is worse at 35% at $X = 16$. Overall, we conclude that: 1) moderate mobility degrades Medley performance only moderately, and 2) Medley continues offering low communication and detection times even if its internal data (ping distances and hence ping probabilities) remain stale.

D. Deployment Evaluation

We implemented a prototype of Medley in the Raspberry Pi (RP) 4 [23] 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. 12(a) and 12(b) show a photograph and a map of one of our topologies. This random topology was in a 6 m × 6 m area. Each device was a Raspberry Pi 4 model B, with 2GB LPDDR4 RAM and Broadcom BCM2711, 1.5 GHz quad-core Cortex-A72 CPU. While Medley works modularly with any ad-hoc routing protocol, for concreteness we use OLSR routing [24] due to its ease of configurability for Pi4s, and popularity in discussion forum posts. 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. 12(b) 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.

1) Failure Detection and Dissemination Latency: From Fig. 13 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. 12) 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 (Section 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.

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 [22]. Fig. 14 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.

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× at m = 3 and 1.5× 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. 15 depicts the bandwidth and FFT for a 600s 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×((N−2)−α)+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 (Section 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 [25]. For this case, 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 [25] 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 route via failed nodes.

IX. RELATED WORK

Classical Failure Detection: Failure detection in datacenters 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 [26]. As described earlier, SWIM [9] is the inverse of heartbeatting, 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 [27] uses flooding and aggregates distributed data at the sink, detecting failure by finding insufficient flow of incoming data. Memento [28] uses a tree for failure monitoring, limiting its scalability under failures. Network-level delays and packet traces can be used for failure detection [29], [30]. 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. A. Asim et al. [31] 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.

REFERENCES

[1] 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, New Delhi, India, 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.” 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. 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 Mobile Comput., vol. 33, pp. 32–49, Dec. 2016.
[5] A. K. Sikder, H. Aksu, and A. S. Ulugaguc, “6thSense: A context-aware sensor-based attack detector for smart devices,” in Proc. USENIX Security, 2017, pp. 397–414.
[6] J. Choi et al., “Detecting and identifying faulty IoT devices in smart home with context extraction,” in Proc. IEEE DSN, 2018, pp. 610–621.
[7] European Parliament, “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,” Off. J. Eur. Union, vol. 59, nos. 1–88, p. 294, 2016.
[8] “Health insurance portability and accountability act of 1996,” ASPE, Chicago, IL, USA, Rep. 104-191, 1996.
[9] A. Das, I. Gupta, and A. Motivallal, “SWIM: Scalable weakly-consistent infection-style process group membership protocol,” in Proc. IEEE DSN, 2002, pp. 303–312.
[10] I. Birman and T. Joseph, “Exploiting virtual synchrony in distributed systems,” in Proc. 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 Proc. WDAG, 1997, pp. 126–140.
[12] R. Van Renesse, Y. Minsky, and M. Hayden, “A Gossip-style failure detection service,” in Proc. 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 Syst. J., vol. 34, no. 2, pp. 414–446, 1995.
[14] I. Gupta, T. D. Chandra, and G. S. Goldszmidt, “On scalable and efficient failure detectors,” in Proc. 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. USENIX OSDI, 2004, pp. 1–6.
[16] K. Driscoll, B. Hall, M. Paulitsch, P. Zumbieg, and H. Sivencrona, “The real Byzantine generals,” in Proc. IEEE DASC, vol. 2, 2004, p. 6.
[17] T. D. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” J. ACM, 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/introtoringpop/
[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] U. Bischoff, M. Strohbach, M. Hazas, and G. Kortuem, “Constraint-based distance estimation in ad-hoc wireless sensor networks,” in Proc. EWSN, 2006, pp. 54–68. [Online]. Available: https://doi.org/10.1007/11669463_7
[22] R. Yang, S. Zhu, Y. Li, and I. Gupta, “Medley: A novel distributed failure detector for IoT networks,” in Proc. Middleware, 2019, pp. 319–331.
[23] “Raspberry Pi 4 model B.” 2016. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[24] OLSR.ORG. “Optimized link state routing protocol.” [Online]. Available: https://tinyurl.com/olsrd-wiki
[25] A. I. Ganesh, A.-M. Kermarrec, and I. Massouli, “Peer-to-peer membership management for Gossip-based protocols,” IEEE Trans. Comput., vol. 52, no. 2, pp. 139–149, Feb. 2003.
[26] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh, “Efficient epidemic-style protocols for reliable and scalable multicast,” in Proc. IEEE SRDS, 2002, pp. 180–189.

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

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

[29] B. Chen, G. Peterson, G. Mainland, and M. Welsh, “Livenet: Using passive monitoring to reconstruct sensor network dynamics,” in Proc. DCOSS, 2008, pp. 79–98.

[30] R. N. Duchê and N. P. Sarwade, “Sensor node failure detection based on round trip delay and paths in WSNs,” IEEE Sensors J., vol. 14, no. 2, pp. 455–464, Feb. 2014.

[31] M. Asim, H. Mokhtar, and M. Merabti, “A fault management architecture for wireless sensor network,” in Proc. IEEE IWCMC, 2008, pp. 779–785.

**Rui Yang** is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Illinois Urbana–Champaign, working with Prof. I. 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 scenarios.

**Jiyu Hu** received the bachelor’s degree in computer engineering from the University of Illinois Urbana–Champaign, in 2021. He is currently pursuing the Master of Computational Data Science degree with the School of Computer Science, Carnegie Mellon University. His research focus was distributed systems, primarily on IoT and edge computing. His area of interest remains in data-intensive systems.

**Jiayan Yang** is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Illinois Urbana–Champaign, working with Prof. I. 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 scenarios.

**Jiangran Wang** received the bachelor’s degree in computer engineering from the University of Illinois Urbana–Champaign, in 2021, where he is currently pursuing the M.S. degree in electrical and computer engineering. His research focus on distributed systems, primarily on the Internet of Things and failure detection algorithms.

**Shichu Zhu** received the M.S. degree in computer science and atmospheric science from the University of Illinois Urbana–Champaign, in 2019. He is currently working with Google Cambridge on Cloud Networking. He has research experience in distributed systems, databases and HCI, as well as the microphysics of cirrus clouds.

**Yifei Li** received the Master of Science degree in computer engineering from the University of Illinois Urbana–Champaign, in 2019. He currently works with Confluent Inc., which is a leading force in event streaming platforms. He primarily works on inter-cluster data replication and cloud services. His prior research focus in distributed systems was around databases and the Internet of Things.

**Indranil Gupta** (Senior Member, IEEE) received the B.Tech. degree from IIT Chennai, in 1998 and the Ph.D. degree from Cornell, in 2004. He is a Professor of Computer Science with the University of Illinois Urbana–Champaign. He leads the DPRG Research Group that works on large-scale distributed systems. He has also worked with Google, IBM Research, and Microsoft Research. He is an ACM Distinguished Scientist.