Memento: A Health Monitoring System for Wireless Sensor Networks

Stanislav Rost
stanrost@csail.mit.edu

Hari Balakrishnan
hari@csail.mit.edu

***Abstract*— Wireless sensor networks are deployed today to monitor the environment, but their own health status is relatively opaque to network administrators, in most cases. Our system, Memento, provides failure detection and symptom alerts, while being frugal in the use of energy and bandwidth. Memento has two parts: an energy-efficient protocol to deliver state summaries, and a distributed failure detector module. The failure detector is robust to packet losses, and attempts to ensure that reports of failure will not exceed a specified false positive rate. We show that distributed monitoring of a subset of well-connected neighbors using a variance-bound based failure detector achieves the lowest rate of false positives, suitable for use in practice. We evaluate our findings using an implementation for the TinyOS platform on the Mica2 motes on a 55-node network, and find that Memento achieves a 80-90% reduction in bandwidth use compared to standard data collection methods.**

I. Introduction

The development of wireless sensor networks has been driven by recent technological advances that have enabled the integration of computing, radio communication, and sensing on tiny devices. Wireless sensornets are now being embedded in our environment for a variety of monitoring tasks [11], [10], [3], [7], [1]. The early successes of real-world deployments has led to a new challenge for researchers: *the management of the sensor network itself*. There are currently few general-purpose tools to monitor the health and performance of deployed sensornets.

There are at least three broad classes of information that a sensornet management system can provide to users and administrators. First, *failure detection*, informing the user about failed nodes. Second, *symptom alerts*, proactively informing the user about symptoms of impending failure or reporting on performance. Third, *ex post facto inspection*, informing the user of the timeline of the events to help infer why a failure or symptom occurred. These classes of information allow users to more effectively debug software, tune parameters for better performance, monitor hardware behavior, provision the wireless network based on offered load, understand why failures occurred, and even prevent failures before they occur.

Designing a sensornet management system involves trade-offs between accuracy, timeliness, and efficiency. The system must not miss too many important events (high detection rate), but yet must not "cry wolf" too often with false alarms (low rate of false positives). Moreover, its reports must be timely, usually within many seconds, rather than hours, of an event. While these goals are desirable generally in monitoring systems in many domains, wireless sensornets impose additional stringent constraints. Because they are often deployed to monitor conditions in remote locations and are expected to run for months or years on small batteries, it is important for a wireless sensornet management system to use energy sparingly. This requirement, in turn, implies that the protocol used to gather information about the health and status of nodes in the network must impose as little communication and processing overhead as possible.

This paper describes the design and implementation of *Memento*, a network management system for wireless sensornets that meets the goals mentioned above. In Memento, the nodes in the network cooperatively monitor one another to implement a distributed node failure detector, a symptom alert protocol, and a logger. The nodes use the Memento protocol, a low-overhead delivery mechanism that attempts to report only changes in the state of each node. This protocol uses existing routing topologies and other protocol's beacons as heartbeat messages, whenever possible.

This paper describes Memento's architecture and protocol (§II), failure detectors (§III), and evaluates their performance on a real-world testbed of 55 sensor nodes (§IV). We show that Memento reduces the communication complexity of monitoring by nearly an order of magnitude compared to the state-of-the-art. Our main results show that a variance-based detector combined with distributed detection can provide timely failure notifications while not exceeding a desired false positive rate. We also address the issue of which other nodes any given node should monitor, and find that loss thresholds provide sufficient control over the tradeoffs in performing this task.

II. Memento Architecture and Protocol

Memento collects the status of all the nodes in the sensor network (numbered 1 through N) in the form of bitmaps endowed with the type semantics of a particular health symptom. In a status bitmap of type t, the k'th bit corresponds to the status of the sensor node whose ID is k. For example, if $t =$"alive", a bit pattern of 1101110 says that nodes 3 and 7 (the "0" bit positions) are not believed to be alive, while the others are. Using type semantics, we can represent any discrete health symptom with bitmaps, given that we can impement a watchdog for that symptom. Health monitoring modules control the bits in such *local* status bitmaps of various types at each of the nodes. Examples of health watchdogs that modify their respective status bitmaps include failure detectors of nodes within the local radio neighborhood ($t =$"alive"), the low battery voltage alarm ($t =$"lowvolt"), local radio congestion ($t =$"congested") etc.

The Memento protocol calculates the *aggregate* result of each node by combining (i.e., bitwise OR'ing) the node's local state with the results of matching type that are produced by its children within the routing topology. Therefore, each result summarizes the status of a node's subtree, including that node. The protocol sweeps the entire network every τ_{sweep}, and delivers the global aggregate result to the *gateway* node. The gateway node relays the information to the a server, which understands the semantics of each bitmap type, and is able to present the information to the network operator in human-readable form.

Memento reuses the main sensing application's routing protocol rather than inventing its own. This approach is well-suited to optimized routing trees commonly used in sensor networks [13].

We observe that, when monitoring many types of node status (such as lists of live neighbors), the data changes infrequently. Other types of health metrics can also be monitored in terms of their crossing of critical thresholds, which also do not change often. An obvious way to leverage this property to save energy is to cache the results of the children at every node, and, whenever no change occurs at a child, reuse the cached results to compute the aggregate result . Once a child synchronizes with its parent, the child can suppress further updates until the synchronization breaks. Nodes become desynchronized whenever (a) the child's result changes; (b) the child node switches its parent in the routing tree; or (c) the parent evicts the child's result from the cache.

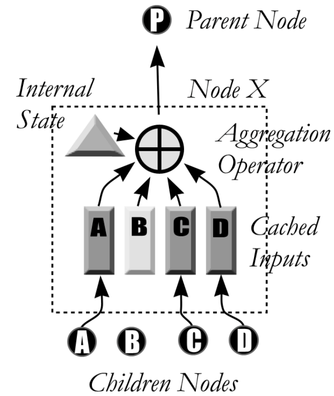


Fig. 1. Diagram of the Memento protocol running on a sensor node X. Child B is synchronized to X, and its result is cached. Nonetheless, updates from A, C, and D change the result of X and prompt it to send an update to its parent P to resynchronize.

The Memento protocol addresses the problems related to maintaining the consistency of the node's result with the parent's cache in the face of packet loss, routing reconfigurations, and node failures. To achieve cache consistency, Memento uses the following modules.

The first module performs *neighborhood and cache management* (NCM), tracks the neighboring nodes, maintains the cache of child inputs, and restricts the attempts by the routing layer to connect to parents whose cache cannot accommodate any more children. Since the majority of traditional routing protocols do not maintain child state or limit the fan-in of the routing topology, we require small modifications to the routing such that NCM can intervene in attempts to connect to new parents, and blacklist some of the candidates for parenthood. A child node's NCM module introduces an extra step in connecting to a new parent, whereby the child explicitly asks the new parent's NCM module to add the results of the former to the input cache of the latter. The parent's module may accept, or reject if the cache is full. Also, the child may consider itself "rejected" after its multiple requests receive no response. If rejected, the child blacklists this prospective parent (i.e., excludes it from consideration until further notice). The NCM lifts the ban only after the previously blacklisted neighbor (a potential parent) announces that it has a free cache slot.

The *synchronization module* assures the coherence between each node and its parent by computing the current result $R_{current}$ from the inputs and its internal state every τ_{sweep}. It also retains R_{sync}, the last result that the parent acknowledged after storing in its cache. For every change of health status affecting $R_{current}$, the synchronization module increments the version number $Ver(R_{current})$. Whenever $R_{current} \neq R_{sync}$, it sends an update containing $\langle Ver(R_{current}), R_{current}\rangle$. The parent must then acknowledge the receipt of this version of the update, and upon receiving this confirmation the child sets its R_{sync} to $R_{current}$.

The third module, the *inconsistency detector*, forces resynchronization in the following four cases:

Child has switched parent. The NCM module of the parent infers this scenario from its child's routing beacons, which contains the ID of the child's current parent. After detecting that a child has switched, the parent frees the child's entry from the cache to avoid using its stale results.

Child has failed. The failure detector (§III) determines a child's failure from heartbeat beacons, and frees the child's cached entry.

Child attaches to new parent. A node's NCM module notifies its inconsistency detector whenever the node attaches to a new parent. The child must then synchronize with the parent to initialize the parent's input cache.

Child evicted by parent. A parent may delete a child's result from its cache because of the parent rebooting, mistakenly deciding that this child has failed, or freeing a cache slot to accommodate another child with very few parent candidates. The child can detect this condition by comparing the parent's result broadcast with its $R_{current}$. Some classes of aggregation operators, like the bitwise OR which Memento uses, allow a child to detect when the result of the parent is missing important information delivered by the child. The problem is that channel loss may cause the child to miss the parent's update and fail to realize its desynchronization. To overcome this, we force each parent to send its result infrequently, after a long period τ_{idle} of silence even when synchronized.

We further optimize the performance of the protocol we describe above. First, Memento can take advantage of the child-parent synchronization and send incremental updates, which are likely to compress better than full updates. To support incremental updates, each child node may keep all of the versions of its results in the range $Ver(R_{sync})..Ver(R_{current})$. The parent may then broadcast a vector of the versions of inputs in its cache as an acknowledgment for updates, or every τ_{idle} when idle. If the parent's current cached version of input from a particular child is $Ver(R_{par})$, then this child can issue an incremental update relative to R_{par}. Second, Memento can perform "lazy" updates. The idea is to suppress the updates if the node believes that sending one will not affect the parent's current result, i.e. in the case when $R_{par} \setminus R_{sync} \oplus R_{current} = R_{par}$. Similarly, we can delay the synchronization with a new parent after the parent switch until the node's result changes or the former parent evicts the result from its cache.

III. Failure Detection in Memento

In this section, we propose several failure detectors. This module monitors the "up/down" status of the node's neighbors within radio range and reports its summary using the Memento protocol. The failure of any node is monitored by a number of other nodes in its vicinity. Failure detection with Memento requires two components: *heartbeats* and a *failure detector*. Each node periodically sends heartbeat messages. A failure detector running on a different node declares a node to have *failed* if a certain amount of time expires since the receipt of that node's last heartbeat.

In this paper, we only deal with fail-stop failures, leaving the issue of Byzantine failures to future work. The schemes we propose in this section guarantee that all failures will eventually be detected. We call the time between the failure of a node and when it is reported to the user the *detection time*.

A failure detector outputs a *liveness bitmap*, which summarizes this node's current belief in the liveness of neighbors. If it considers a node k alive, it sets the k^{th} bit of its liveness bitmap $live_{local}$ to 1. Otherwise, the $live_{local}[k]$ is set to 0. The Memento protocol carries such liveness bitmaps to the gateway, aggregating them along the way, and delivering the final result to the Memento front-end. The front-end can then compare the list of live nodes with the roster of deployment to determine which of the nodes have failed.

Each node listens to *heartbeats* from a subset of its neighbors in the network.[1] The failure detection module could send its own heartbeats, but we advocate reusing the broadcasts of other periodic protocols that might already be running (e.g., routing advertisements, time synchronization beacons, sensor samples, etc.). The heartbeat protocol must be periodic and must include the identifier of the node sending the heartbeat. We denote the average time between heartbeats as τ_{hb}. These heartbeats are not related to the time period, τ_{sweep}, over which the Memento protocol gathers status information.

A. Failure Detectors

Our initial attempt at designing a failure detector mimics the behavior of the state-of-the-art approaches to failure detection (such as Sympathy [8], or schemes based on TinyDB [6]), in which the gateway interprets the lack of arrival of data from a particular node within a fixed time period as an indication of its failure.

This detector, which we call **Direct-Heartbeat**, takes advantage of the periodicity of network sweeps. After every sweep, it resets $live_{local}$ to all 0's. Whenever a heartbeat from a node X arrives, the failure detector set $live_{local}[X]$ to 1. As long as a neighbor manages to get one heartbeat per τ_{sweep} across, Direct-Heartbeat will consider it alive. When $\tau_{hb} << \tau_{sweep}$, this scheme achieves low false positives.

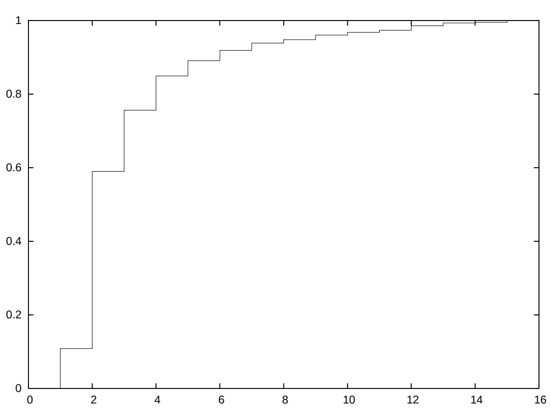


Fig. 2. The average of empirical cumulative distributions of the number of consecutive heartbeats lost in transmission from neighbor to neighbor, across all the nodes in our sensor testbed, over one day. No node had failed during this experiment.

Because Direct-Heartbeat does not adapt to wireless packet losses, it performs poorly in practice. Figure 2 shows the cumulative distribution of the lengths of gaps of incoming heartbeat packets from each neighbor for one experiment in our sensor network testbed (an in-building 55-node network). On average, a neighbor's heartbeat stream will miss two consecutive heartbeats in a row, but in 10% of all cases nodes will miss six in a row, and 5% of the time, ten in a row. Meeting a desired false positive rate of, say, 1%, with Direct-Heartbeat in this scenario would require setting the ratio of heartbeat frequency to the frequency of sweeps high enough to accommodate the worst loss of any of the links in the routing topology (in this case, 16 heartbeats per sweep). That can be achieved either by increasing the rate of heartbeats to be very high, or by making the detection time very high because of the longer sweep period.

To overcome this shortcoming, the failure detector must adapt to the incoming packet loss. In particular, we need to estimate how many heartbeats from a particular neighbor must be lost in a row to indicate its failure. The **Variance-Bound** failure detector maintains the mean and standard deviation of the number of consecutively missing heartbeats that are typical of each live neighbor. This failure detector takes one additional input parameter from the user: the maximum target rate of false positives, FP_{req}. It then attempts to guarantee this rate in estimating a bound on the maximum number of consecutive heartbeats which may be missed by each neighbor while it is still alive. We call this bound a *heartbeat timeout*.

Variance-Bound calculates the timeout using a single-tailed variant of Chebyshev's inequality: $P\left[(X-\overline{X}) \geq t \cdot \sigma_X\right] \leq \frac{1}{1+t^2}$.[2] This formula expresses a bound on the distance of a random variable X from its mean $\overline{X}$ in terms of a multiple t of its standard deviation σ_X. Suppose that G_i denotes heartbeat gaps (i.e., the number of consecutively missed heartbeats from neighbor i), with $\overline{G_i}$ and σ_i being its mean and variance. Thus, to attain FP_{req}, we can derive the *heartbeat timeout*, HTO_i, for each monitored neighbor i from Chebyshev's inequality as follows:

$$HTO_i = \overline{G_i} + \sigma_i \cdot \sqrt{\frac{1-FP_{req}}{FP_{req}}}$$

Because Chebyshev's inequality holds for all distributions, it is able to always guarantee the requisite false

[1] We investigate the question of which nodes should monitor any given node later in this section.

[2] We can derive a version with a tighter bound for unimodal distributions of heartbeat gaps, but per-neighbor distributions (unlike the aggregate in Figure 2) are not necessarily uni-modal.

positive rate (and in practice, as long as the distribution does not change suddenly). On the other hand, to achieve that guarantee it is known to provide loose bounds, which might lead to overly long detection times.

To reduce detection times, we investigate a non-parametric failure detector, which we call **Empirical-CDF**. This detector maintains a compact representation of an empirical probability distribution function (PDF) of gap durations. Whenever the failure detector receives a heartbeat from a monitored neighbor, i, whose G_j prior heartbeats were lost, it updates the PDF vector: $PDF_i[G_j] = PDF_i[G_j] + 1$. Using this representation, the probability of encountering a lapse of length G_i is $\frac{PDF[G_i]}{\sum_j PDF[j]}$.

Combining the PDFs for each neighbor results in an empirical CDF of their gap durations. The CDF characterizes $P[G_i < X_i]$, the probability of the duration being less than some X_i for each neighbor i. If we want to assure a 5% FP, HTO_i has to be set to a value which has a smaller than a 5% chance of occurring. The failure detector can determine the HTO_i from the complement of the CDF, by searching for the the minimum HTO_i such that the probability of missing HTO_i or more heartbeats is smaller than the false positive parameter ($P[G_i \geq HTO_i] < FP_{req}$):

$$\left\{ \min HTO s.t. \frac{\sum_{j=0}^{j=HTO} PDF[j]}{\sum_{k=0}^{k=len(PDF)} PDF[k]} \geq (1 - FP_{req}) \right\}$$

Empirical-CDF must seed its model with a number of initial observations to be statistically representative. Otherwise, the first new samples of highest lapse lengths will count as false positives. Hence, for the first N_{CDF_init} (10 in our experiments) samples, we use a rough estimate of HTO_i based on the empirical mean which we calculate from the PDF vector.

As time passes, the probability density model may become unrepresentative of the current wireless network conditions. Additionally, the values in some bins may grow to exceed the precision of the data structures. To solve these problems, Empirical-CDF decays the PDF vector every τ_{CDF_scale} by scaling the incidence counters in its bins with a decay constant g, $0 < g < 1$. In our experiments, we do not apply such scaling because they are too short-lived.

B. Neighborhood Monitoring and Opportunism

The minimum subset of neighbors that each node must monitor includes its children in the routing topology. This coverage is necessary for the correct operation of the Memento protocol, which needs to be able to invalidate a failed child's cache to avoid basing its parent's result on stale input.

However, if the resource budget affords it, opportunistically monitoring other nodes in the radio neighborhood may provide more robustness against loss and topology reconfiguration. Packet losses among the wireless receivers of a heartbeat packet broadcast may be uncorrelated. If a parent node misses a long train of heartbeats, other neighbors may receive a large enough fraction of these packets to override the failure opinion of the parent in aggregation along the way to the gateway.

To monitor multiple neighbors, Memento maintains a supplementary bitmap, $live_{opport}$, containing the bits which denote the liveness of "well-connected" neighbors (i.e., those with incoming loss less than $lossThresh$, e.g., 50%). Memento treats $live_{opport}$ similarly to an input from a child, aggregating it with inputs from its children and $live_{local}$ in computing the aggregate result of the current sweep. Setting $lossThresh$ to admit high-loss neighbors may cause problems, as we discuss in Section IV-E.

C. Detecting Network Partitioning

A temporary network partition may occur when a node fails, because all its descendants must wait until the routing layer connects them to new parents. Also, a persistent partition may occur in topologies with insufficient redundancy after a node failure. While partitions fall under our definition of failure, it would be useful to be able to infer them.

Our proposal for detecting network partitions is to use two additional bitmap types: $failureTip$ and $partitioned$. Both bitmaps are sent by the immediate parent of the failed node, and aggregated only by their ancestors. The $failureTip$ bitmap specifies the IDs of failed nodes. The $partitioned$ bitmap aggregates the IDs of the descendants of the failed nodes, derived from the failed nodes' results extracted from the caches of the parents of the failed nodes. The gateway may determine the set of partitioned nodes using the set arithmetic expression $partitioned - (failureTip - partitioned)$.

IV. Implementation and Evaluation

A. Platform and Testbed

We have implemented the Memento protocol and failure detection module TinyOS [5] on the Mica2 mote platform. In a typical configuration, the Memento protocol and the failure detection module use less than 400 bytes of RAM, which is only 10% of the total memory on the Mica2 platform.

We conducted experiments on a real-world testbed to answer several questions. First, we investigate the performance of the Memento protocol as a function of the stability of routing and the rate of change of the results it reports. Second, we compare the performance of the different failure detectors. Finally, we investigate the tradeoffs in choosing a subset of neighbors to monitor opportunistically.

Our experiments use a 55-node in-building wireless network testbed of Mica2 Mote sensor nodes. All nodes are attached to the Ethernet reprogramming boards, and use a wired serial channel for collecting results.

We implemented the ETX [2] routing protocol in this network. The protocol's routing beacons also serve as node heartbeats. Each experiment is 45 minutes long, and Memento sweeps the state of the network every τ_{sweep} = 30 seconds.

B. Performance of Memento

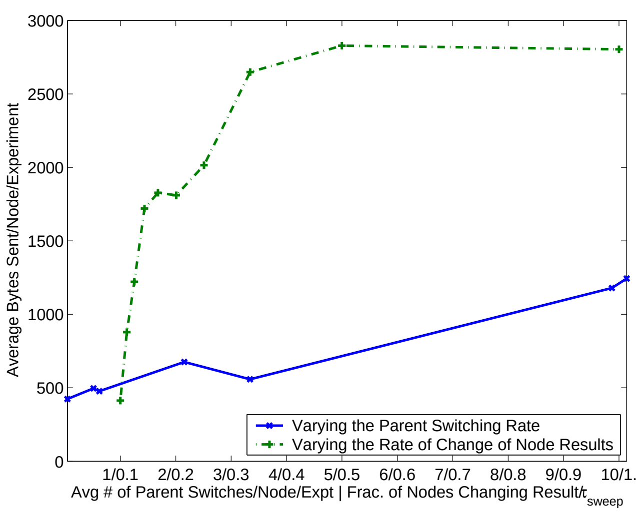


Fig. 3. The effect of the rate of switching parents or sending updates on communication overhead.

We evaluate the impact of routing stability by varying ϵ_{sw}, the parent switching threshold of ETX routing. In this scheme, a node limits the set of prospective parents to nodes whose ETX metric is smaller than the current parent's by more than ϵ_{sw} (the units are packet transmissions). Increasing ϵ_{sw} reduces the likelihood of switching the parent, and we vary it between 0 and 4 in increments of 0.5. We induce no failures during the experiment. The standard deviation of the plots is within 15% of the mean.

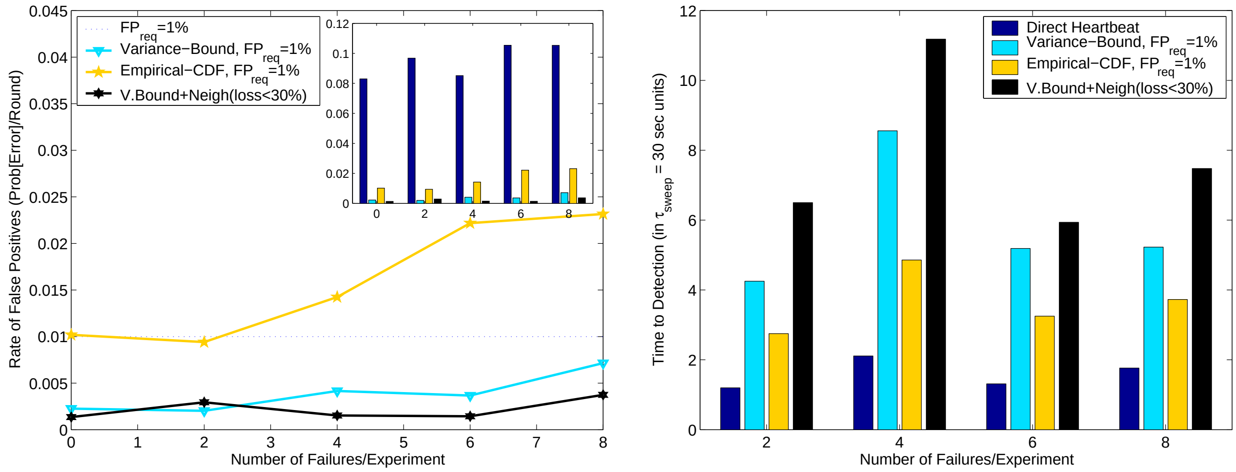
When the threshold is 0, each node will switch to the "best" of the neighbors, even if it is only marginally better than the current parent. With this setting, the routing topology becomes very volatile, with every node switching parents ≈ 10 times during the experiment. For $\epsilon_{sw} > 3$, the nodes switch only once.

The solid line plot in Figure 3 shows the effect of changing ϵ_{sw} (we show the number of parent switches per experiment), and the resulting frequency of parent switches on the amount of update traffic generated by Memento. Our results show that the amount of communication grows proportionally with the number of parent switches. We note that even when routing is most volatile, the amount of communication is only three times worse than the most stable setting.

The second part of the evaluation performs the opposite of the above: we fix the routing topology, and evaluate how randomly changing the result of each node at a period between changes, α, affects the bandwidth requirements. Varying α between 30 and 300 seconds in increments of 10, we find that the efficiency of Memento depends mostly on the number of the network nodes that send updates every round.

In Figure 3, the dashed plot illustrates that, in the routing topologies specific to our deployment, increasing the fraction of sources that spur chains of updates that propagate towards the gateway makes bandwidth requirements grow rapidly. The increase in the curve is quite sharp because increasingly many intermediate nodes must resynchronize with their parents to push updates to the gateway. In the worst case, each node will send one update per round.

In our analysis of Memento's performance (omitted for want of space) we have determined that, given randomly picked sources of updates, it performs best in "short, wide" balanced routing tree as opposed to "long, narrow" imbalanced trees. In "short, wide" trees fewer intermediate nodes get involved in relaying other source's updates to the gateway, and the impact of increasing the rate of change of results on the bandwidth



(a) Probability of mislabeling a live node as dead. Inset: Direct Heartbeat generates a large number of false positives.

(b) Delay between a failure and its detection.

Fig. 4. Evaluation of the failure detector performance.

is more gradual than in Figure 3.

C. Failure Detection Experiments

This section evaluates and compares the performance of Memento's failure detectors. We focus on the false positive rate and detection time of the detectors under various conditions, and also report the number of bytes transmitted in each experiment.

Our experiments mimic the anticipated real-world use of the failure detector module, except that we drastically scale down the timescales of protocol periods in order to reduce the total running time of each experiment. We use the ETX routing protocol, which sends its routing and time synchronization beacons (serving double duty as heartbeats) every τ_{hb} = 10 seconds. τ_{sweep} = 30 seconds. We set the parent switching threshold ϵ_{sw} to 1.5. In each experiment, we choose k random nodes for failure, and, for each of them, a random time of failure instant. Nodes simulate failure by ceasing communication.

Figures 4(a) and 4(b) show the results of these experiments. Each sample point is an average of nine trials. The routing topology for each trial may vary, but the failure schedules are identical across the failure detectors in each trial.

We can clearly see in the plots that Variance-Bound is able to meet the desired false positive rate, even doing better than required. That result is heartening because false positives could occur for many reasons in practice, including slow routing convergence while switching away from a failed parent, or a lag in synchronization between a child and the parent cache while the former is attaching to the latter, or because each node stops sending updates to its parent after three unacknowledged retransmissions (the node will try again during the next sweep). In practice, the scheme is able to handle these situations most of the time.

The factors stated above also explain the overall trends of false positive rates growing with the number of failures (Figure 4(a)). When no failures occur, the false positives are caused by normal routing optimization in response to fluctuations in loss. As we ramp up the failure rate, however, an increasing number of descendants of failed nodes end up temporarily disconnected from the gateway. Any sweeps that occur during the delay until they connect to new parents cause false positives.

The detection time is also affected whenever, for every actual failure, the factors we list above delay its discovery. Instead of looking at the absolute impact of these factors, we instead assume they affect all the failure detectors uniformly, and consider the differences between the schemes in Section IV-D.

Another set of results (not shown to conserve space) shows that nodes running Direct-Heartbeat send between 3150 and 3300 bytes per experiment, while Memento-based approaches consistently require only between 320 and 500 bytes of transmissions per experiment. Moreover, the amount of communication does not grow appreciably with the number of failures. The reason is that the sequences of the updates generated by each failure event are comparable in the volume of traffic to the updates Memento issues in steady-state to keep the caches synchronized in the face of routing changes. The latter, in conjunction with initial synchronization traffic ($\approx$ 150 bytes) and maintenance traffic such as acknowledgments, also explains why communication costs are not zero in steady-state.

D. Comparing the Failure Detectors

The Direct-Heartbeat failure detector reports that a neighbor is alive only after receiving one or more of its heartbeats since the previous network sweep. This scheme is representative of the commonly used approaches which rely on fixed-length failure timeouts. Direct-Heartbeat has an unacceptably high rate of false positives, between 8.2% in a network with no failures and 10.6% when eight nodes fail per experiment (Figure 4(a), inset). Such poor performance is due to $\frac{\tau_{sweep}}{\tau_{hb}}$ not being very large. Since heartbeats are broadcast unreliably, it is quite likely to lose three consecutive heartbeats from the same neighbor (resulting in failure opinion for that node), or to fail in transmitting updates to parents in three retransmissions or less.

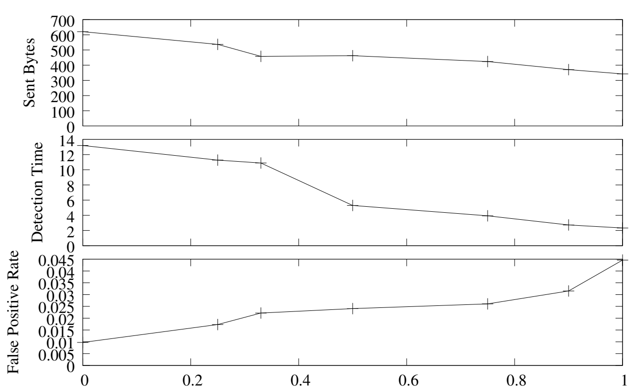


Fig. 5. Performance of Variance-Bound ($FP_{req} = 1\%$, $\epsilon_{sw} = 1.5$) as the threshold on incoming loss (on the X axis) limits the subset of monitored neighbors. Each node always monitors its children, and also monitors all neighbors whose loss rate to the node is not greater than X.

The Empirical-CDF failure detector shows a vast improvement over Direct-Heartbeat. It is able to meet the 1% false positive rate requirement when no failures occur, but not otherwise. This scheme's timeout bound is determined by prior observations of gaps in a neighbor's heartbeats, which trims the gap distribution's tail. However, the maximum timeout possible is the longest previously observed gap, and Empirical-CDF produces a false positive every time a neighbor's gap is longer than any prior samples. The chance of a false positive is especially high in the early phases of connecting to a parent, when the CDF is not very representative. Failures of parents are likely to cause widespread migrations of descendants to new parents, and Empirical-CDF simply does not learn about them quickly enough to accommodate their variance, which explains the growth of its failure positives.

Variance-Bound is the best performer, providing a false positive rates of 0.22% to 0.71%, well below the goal, at the expense of 57% longer detection times, relative to Empirical-CDF. At the experiment's timescales, the delay does not seem significant, but as we inflate the periods of protocols to realistic durations, it could translate to much longer periods of undetected failures, on the scale of days.

E. Performance of Opportunistic Monitoring

Figure 4(a) shows that the performance of Variance-Bound can be further improved by monitoring a bigger subset of the neighbors with good connectivity (whose incoming loss is $< 30\%$).

However, given the constrained resource budget of the sensor nodes, it may be impossible to monitor all neighbors. More important is the question of how the choice of the subset of neighbors to monitor would affect the performance of failure detectors.

The graphs in Figure 5 aggregate the results for various scopes of opportunism. When $X = 0$, nodes keep track of all their neighbors, and when $X = 1$, just the children. In general, all other neighbors whose loss is greater than the fraction along the X axis are rejected.

Our results show that rampant opportunism reduces the false positive rate significantly, because the more neighbors track a given node, the more paths to the gateway are likely to carry its status. However, tracking all neighbors inflates the detection time by a factor of six, and causes twice as many transmissions of updates relative to tracking just the children.

The sharp increase in the detection time results from monitoring neighbors whose heartbeats are unreliable. High packet loss leads to inflated heartbeat timeouts, which may cause a node to maintain that its dead neighbors are alive long after their failures.

The increases in transmission rate are caused by each node's result changing more frequently. That is because

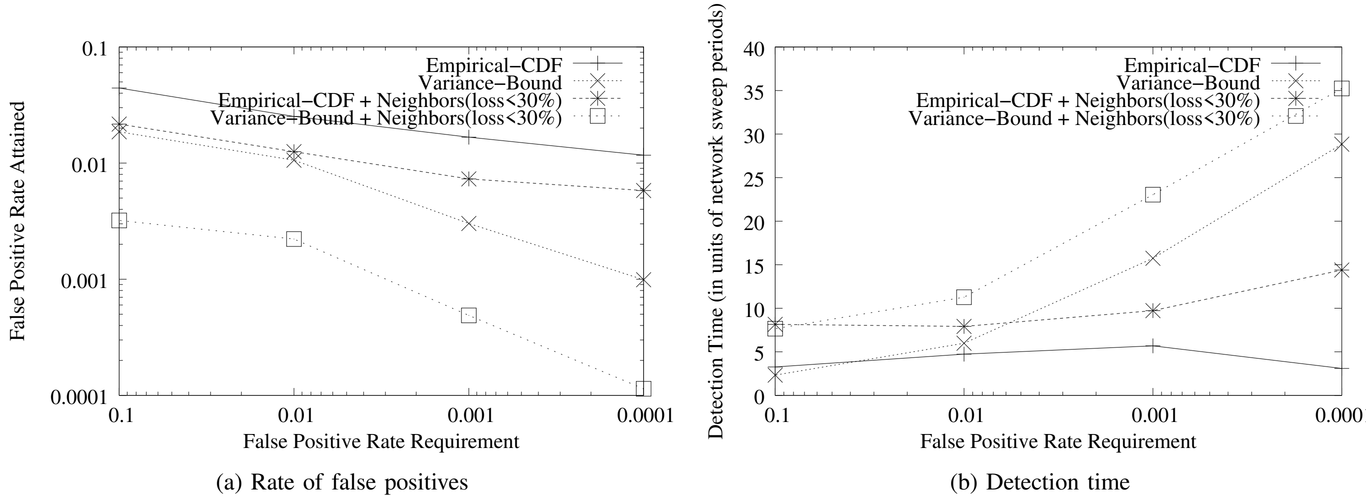


Fig. 6. Performance of failure detectors as target false positive rate grows stringent.

more bits in their liveness bitmaps actively track their neighbors' status, and are subject to change.

An interesting feature of this graph is the sudden drop in the detection time that occurs between the admission thresholds of 33% and 50%. We determined that this drop occurs as soon as the neighbors on the "far" side of the routing tree, having the highest variance of packet loss, are disqualified as parent candidates.

The results lead us to believe that, in applications where the false positive rate is most important, transmitting 70% more traffic achieves a four-fold improvement in that metric. In this case, it may be a worthwhile tradeoff and the network operator should consider monitoring all neighbors.

F. The Limits on the False Positive Rate

We would like to minimize the incidence of false failure reports, which may drive network operators to perform unnecessary and costly maintenance. This metric compounds with both the size of the network and the passage of time, so it is important to determine its limits.

To explore this dimension of the performance, we vary the FP_{req} from 0.1 down to 0.0001 in factors of 10. We evaluate whether our schemes are able to meet the target requirement across the experiments with varying failure rates, from two to eight failures per experiment.

Figure 6 shows the results. Empirical-CDF cannot meet the 1% requirement, and even its neighborhood-opportunistic version can barely attain this goal. Variance-Bound's best performance brings it close to meeting the 1% guarantee. Only by monitoring additional neighbors (those whose heartbeat loss is less than 30%) can this scheme achieve the four nines requirement.

Such performance increases detection time considerably. The Figure 6(b) shows that detection timeouts grow by a factor of 4-5 in order to meet the false positive target that is five orders of magnitude lower.

The fundamental reason for Empirical-CDF's lackluster performance is because it takes too long for it to learn a representative model of heartbeat gaps. Nodes emit 270 heartbeats in the course of each experiment, which limits the maximum number of gap samples in the CDF to 134. With this resolution, achieving less than 1% goal rate becomes infeasible. In fact, nodes collect ≈ 26 heartbeat gap samples on average, per experiment. While it is possible that the performance of Empirical-CDF will improve over longer deployments, it is difficult to predict when the probability density represents an accurate estimate of the loss process, and network operators may not have the patience to wait that long.

V. Related Work

Our approach builds on TinyAggregation [6], which highlights the communication savings resulting from aggregation operators. The design of Memento is related to TiNA [9], a proposal in which nodes suppress their transmissions if their result is within tolerance value of their last result. Our protocol improves on TiNA by robustly handling network reconfigurations and failures, and implementing incremental updates.

Sympathy [8] logs communication statistics and attempts to identify and localize node failures. The system samples the neighbor table, packet counts, uptime and congestion and periodically sends them to the gateway. The network user can then infer about the cause of the failure from the metrics, and classify the problem as one from a pre-determined list of network-related causes. While this system classifies the cause of the failure, to a limited extent, it is not bandwidth-efficient. Additionally, Sympathy is similar to Direct-Heartbeat in its design, and could benefit from a Variance-Bound detector design.

Failure detection based on random gossiping [12] can assure a specific rate of failure, but suffers from inherent flaws of gossip protocols, such as slow initialization and very long detection time. In a network of 50 nodes, it requires over 30 rounds to achieve $FP_{req} = 1$. More importantly, this work does not deal with variable packet loss rates, common in sensor networks.

Another randomized failure detector balances the communication load across nodes [4]. This protocol pings a randomly selected neighbor, and if it does not respond, then pings it through a subset of neighbors. While this protocol can be tuned to achieve a specific false positive rate, its bandwidth requirements grow dramatically in the presence of packet loss.

Recent work on failure detectors in overlay networks [14] discusses a number of approaches. Using a probe-and-ack mechanism to ascertain neighbor liveness, nodes share information to reinforce their opinions regarding the liveness status of neighbors. In contrast to our work, the failure detectors proposed in [14] are designed for point-to-point links, and offer no guarantees on the rate of false positives.

VI. Conclusion

This paper makes four main contributions in the area of sensor network management. First, Memento demonstrates that taking advantage of status invariance saves bandwidth and energy. The Memento protocol consumes nearly an order-of-magnitude less bandwidth relative to state-of-the-art approaches that transmit status messages with fixed periodicity.

Second, we find that monitoring more neighbors does not lead to better performance. The communication costs of involving more neighbors and the impact of high-loss neighbors on detection time suffer disproportionately to the improvement in accuracy. However, constraining the monitoring scope to a few well-connected neighbors provides good detection times and false positive rates.

Third, we show that even in indoor environments, the use of neighborhood opportunism and monitoring redundancy is required to achieve practically acceptable false positive rates.

Finally, our evaluation allows us to make recommendations on the failure detector to use depending on the application requirements. If detection time if of primary importance but sensor samples must not be missed, then the Empirical-CDF method would be preferable because it trims the tail of the heartbeat timeout model. On the other hand, if certainty in the failure opinion is at premium, then the Variance-Bound technique in conjunction with neighborhood opportunism would be preferable.

References

[1] http://www.archrock.com.

[2] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom*, San Diego, CA, September 2003.

[3] http://www.ember.com.

[4] I. Gupta, T. Chandra, and G. Goldszmidt. On scalable and efficient distributed failure detectors. In *PODC*, Newport, RI, August 2001.

[5] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler. The emergence of networking abstractions and techniques in TinyOS. In *NSDI*, March, San Francisco, CA 2004.

[6] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, Boston, MA, December 2002.

[7] http://www.millenial.net.

[8] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *SenSys*, San Diego, CA, November 2005.

[9] M. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *MobiDE*, San Diego, CA, September 2003.

[10] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. In *SenSys*, Baltimore, MD, November 2004.

[11] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson, and D. Culler. A macroscope in the redwoods. In *SenSys*, San Diego, CA, November 2005.

[12] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middleware*, The Lake District, England, September 1998.

[13] A. Woo. *A Holistic Approach to Multihop Routing in Sensor Networks*. PhD thesis, UC Berkeley, 2004.

[14] S. Zhuang, D. Geels, I. Stoica, and R. Katz. On failure detection algorithms in overlay networks. In *INFOCOM*, Miami, FL, March 2005.