(Poster Abstract) PerDB: Performance Debugging for Wireless Sensor Networks

Veljko Pejovic
University of California, Santa Barbara
Email: veljko@cs.ucsb.edu

Cormac J. Sreenan
University College Cork, Ireland
Email: cjs@cs.ucc.ie

***Abstract*—The characteristics of wireless sensor networks are such that traditional approaches to network management and monitoring are inappropriate. In this paper a system for performance debugging in wireless sensor networks is presented. It tracks two key metrics - the delay and reliability of message delivery. Its principle of operation is to reactively isolate and diagnose problems once they occur, rather than proactively probe for data in order to provide real time monitoring - an approach that would be less resource efficient and would interfere with normal traffic flows. The system passively gathers information from the existing application traffic and reacts upon signs of problems, selectively probing misbehaving areas until the causes are determined and reported. The paper gives an outline of our performance debugging system PerDB along with preliminary results.**

I. Introduction

Debugging a wireless sensor network is not a straightforward process, and while there has been some work on software debugging, network management and monitoring (e.g. [2],[3],[1]) there has been little attention to *performance* debugging of a WSN. Our goal is a performance-focused debugging system for wireless sensor networks, emphasizing the difference between problems that manifest in performance failures as opposed to functional failures. Obviously the determination as to whether a network is operating within or outside acceptable performance bounds must be application-specific. Our aim is not to design a system that will provide a complete picture of the network state; rather, we strive towards debugging - recognizing faulty situations, localizing them, and finding out the true causes of the undesired events. The contribution of this paper is to present a performance debugging system that passively monitors a WSN in order to detect when it is operating outside of the given thresholds for message delivery latency and reliability, and then uses selective probing to localize and identify the root causes.

II. Approach

The focus of measurement in the proposed system is message delivery, which represents the fundamental service provided by a WSN. While message delivery can be characterized by several metrics, the key generic metrics of interest to most applications are clearly the delivery latency and delivery reliability, both observed from end-to-end. These two metrics encompass a wide range of applications that require performance assurances for message delivery in a WSN.

Our basic approach to measurement is passive, and recognizes that with small modifications in place, the existing traffic in a WSN can be used to piggyback information useful for detecting and locating problems. In contrast to an active, probing-based, approach, the use of a passive approach has significant benefits for energy consumption and is less intrusive. Yet, for the successful isolation of the problems the use of probing cannot be avoided. When probing, we adopt an approach in which nodes aim to maximize to use of space to include as much debugging information in the replies as possible, so that a minimal number of probes is sent and no significant interference with the data traffic is introduced.

III. System Design

In PerDB each packet gathers key performance information as it travels from source to sink. Delay information is added to the cumulative delay field contained in the packet payload. In a similar manner, aggregate packet retransmissions are counted on each of the nodes, together with the total hop count a packet takes to reach the sink. In our implementation for TinyOS we use an additional fours-bytes for carrying this information.

Protocol: Although this approach results in relatively little additional overhead, it is not always feasible to isolate errors via analyzing the relevant information in just a few bytes of each message. In that case the sink relies on a request-response protocol to probe the network. It can probe a specific node in the network, eliciting a response message that is used to gather more detailed performance information about the path. The node's reply consists of information similar to that described above; however, the fact that the entire message can be used allows detailed information for each hop along the path to be obtained. Information such as a node's number of neighbors, battery levels, sources of delay and others can be harvested as well. The probing process consists of the following steps: (a) A probe is sent from the sink towards a problematic node; (b) Each node on the probe's path is "told" what to measure, and how long to wait for its child's answer; a timeout value is set from the moment a probe visits a node and is directly proportional to the number of nodes yet to be visited by a probe; (c) If it gets a probe reply from the child, a node combines its own reply with it and sends to the sink; (d) Otherwise if after the timeout the node does not receive the

answer from its child it generates its own answer and sends it to the sink.

Algorithm: After the data arrives to the sink it is analyzed according to the application's needs. Consider the basic metrics: a network does not perform well if the amount of delivered data is less than expected, or if the average traffic delay is above a certain threshold.

Since the system assumes a simple tree topology it may be able to localize clusters of nodes that are experiencing high delay, in that case the probes are sent towards the clusters' heads since all delayed traffic must go via these common-parent nodes. On the other hand, the paths experiencing the delay may not be under a shared parent node, but we can still try to find some overlapping of the delayed paths along the way; in that case the probes are sent to the overlapped parts first. If there is no overlap then every single node is probed.

In comparing paths, the system uses information derived from "healthy" traffic, i.e. traffic delivered without severe delay and without per-hop retransmissions. Overlapping between a path that a packet which was not delayed and the one which was delayed took can tell us where to look for errors, so that probes are not sent to explore links that belong to the overlapped problematic paths. However, if "healthy" data traffic cannot be utilized then every node whose packets are delayed may need to be probed. This is optimized by probing nodes on the most important paths - the ones whose links are used by the largest number of nodes in the network. It is more likely that fixing errors along these lines will allow enough nodes to function properly.

PerDB deals with the problem of low traffic or an absence of it by probing clusters that are furthest from the sink.

IV. Experimental Evaluation

In order to conduct a preliminary evaluation of PerDB we modified TOSSIM source code to enable the necessary features our system uses. On top, a debugging layer was built above which a stub application is running. We focused on four aspects: (1) time to debug as a function of number of nodes, (2) benefit of using probing in terms of correctly identifying probematic links, (3) impact of number of probe packets on correctness and (4) time to debug relationship with average node degree in topology. Due to space restrictions we only show the proof-of-concept results in this paper.

We argue that even three bytes reserved for debugging purposes in each of the data messages can substantially improve the search for problems, pointing out to problematic areas; otherwise the probes are sent randomly to different parts of the network, in a greedy manner, trying to cover as many nodes as possible. In the experiments the number of nodes is incrementally increased, while keeping the same ratio of errors (lossy links, congested areas) versus well performing links. A comparison is made between the time needed for a system with and without piggybacked debug data to isolate and correctly determine the types of errors. As shown in figure 1, greedy probing does not scale - once the network size gets larger, it becomes much faster to probe the problematic areas

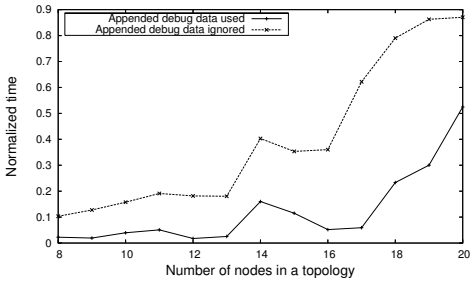

Fig. 1: Normalized debugging time as a function of number of nodes

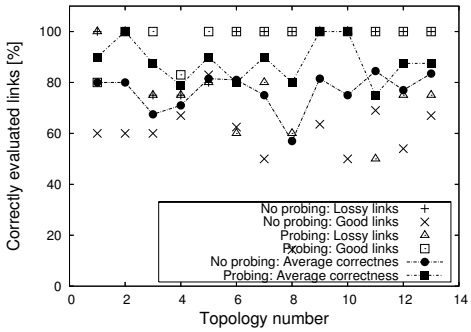

Fig. 2: Percentage of correctly evaluated links for the two approaches

only. It is worth noting that the additional negative effects of probing such as interference with the regular traffic were not considered. However, the probing is still necessary in a number of cases as the piggybacked debug information is insufficient for successful problem isolation. We examined a set of different (in size, ordering of nodes and placement of erroneous links) topologies and compared the correctness of reporting from the system that does not probe and the one that probes if application specific performance thresholds are exceeded. More precise diagnosis is observed when probing is in place (Figure 2). Without probing it is hard to conclude about the source and location of problems. Naturally a tradeoff exists regarding the speed and accuracy of diagnosing performance violations versus the energy required for probing.

V. Conclusion

We have presented a concise summary of PerDB, a performance debugging protocol for wireless sensor networks. PerDB uses a passive approach to gather information about network performance, together with probing when necessary. Ongoing work is to extend PerDB with a more sophisticated approach to inferring sources of performance violations and to deploy on our sensor node network management testbed.

References

[1] W. L. Lee, A. Datta, and R. Cardell-Oliver. *Handbook on Mobile Ad Hoc and Pervasive Communications*, chapter Network Management in Wireless Sensor Networks. American Scientific Publishers, 2007.

[2] Nithya Ramanathan, Eddie Kohler, and Deborah Estrin. Towards a Debugging System for Sensor Networks. *International Journal of Network Management*, 15(4):223–234, July 2005.

[3] M. Ringwald and K. Romer. SNIF: A Comprehensive Tool for Passive Inspection of Sensor Networks. In *GI/ITG KuVS Fachgesprach Sensornetze*, Aachen, Germany, July 2007.