Improving Network Link Quality in Embedded Wireless Systems

Andrew R. Dalton, Jason O. Hallstrom
Clemson University
{adalton,jasonoh}@cs.clemson.edu

Hamza A. Zia, Nigamanth Sridhar
Cleveland State University
{n.sridhar1,h.zia}@csuohio.edu

Abstract

Embedded wireless networks are finding increased adoption across a range of application domains. Unfortunately, these systems are notoriously difficult to deploy at scale; system reliability tends to degrade unexpectedly. This difficulty is due, in part, to link quality variations that impact application correctness and performance.

In this paper, we present and evaluate an alternative radio stack implementation for TinyOS*, a popular operating system for embedded sensor networks. The radio stack is designed to improve the reliability of middle- to high-quality network links, while identifying and avoiding transmissions over low-quality links. The implementation is evaluated in the context of two representative routing scenarios: (i) neighbor-to-neighbor and (ii) distributed convergecast.*

1 Introduction

Embedded wireless systems are gaining wide adoption across diverse application domains. The emergence and integration of small, low-cost, low-powered sensors is one of the principal contributing factors. As a result, embedded *sensor* systems constitute one of the fastest growing system categories, with applications ranging from volcanic activity monitoring [12], to structural damage detection [2], to wildfire prediction and tracking [8, 4] — and many others. One experiential result emerging from the development of these systems is that achieving high reliability in large-scale deployments is notoriously difficult.

One important factor contributing to this difficulty is the unusually high degree of link quality variation found in these systems. The variation is often significant, resulting from geographic node distribution, environmental factors (*e.g.*, physical obstructions), and temporal phenomenon (*e.g.*, increased network load, periodic external interference). As a result, designers are forced to contend with links that exhibit both persistent and transient message loss. Coupled with the highly concurrent and distributed nature of embedded sensor systems, accounting for the myriad of fault scenarios resulting from link failures can be overwhelming. Consequently, the resulting systems, when deployed at scale, tend to exhibit unexpected behaviors and performance characteristics.

In this paper, we present an alternative radio stack implementation for TinyOS [9], the defacto standard operating system for embedded sensor networks. The radio stack implementation is designed to limit link quality variation, improving the reliability of mid-to-high-quality links, and identifying — and avoiding transmissions over — low-quality links. The main contribution of the paper is a detailed evaluation of the reliability benefits provided by the solution. The study is performed on a physical network testbed consisting of 80 wireless sensor nodes. Two representative routing scenarios are considered: (*i*) neighbor-to-neighbor and (*ii*) distributed convergecast. The first scenario corresponds to basic one-hop neighborhood communication. The second is generally used for distributed data exfiltration to upper-tier base station nodes.

Paper Organization. Section 2 presents an overview of the radio stack design. Section 3 summarizes the testbed infrastructure used to evaluate the implementation. Section 4 describes the experimental setup for each of the evaluation scenarios, and presents the experimental results. Section 5 provides a brief overview of related work. Section 6 concludes with a summary of contributions.

2 Design Overview

The alternative radio stack implementation is designed as a reusable component for TinyOS. The component, `ReliableComm`, serves as a drop-in replacement for `GenericComm`, the default radio stack component distributed as part of the operating system. The new component is implemented as a *wrapper* over `GenericComm` using the *Decorator* pattern [7] for TinyOS. (The pattern is a variant of the standard pattern of the same name originally characterized in [6].) Hence, `ReliableComm` supports the same software interfaces as `GenericComm`, providing functions to enable applications to transmit and receive messages over the wireless radio. As a *wrapper*, `ReliableComm` maintains a reference to `GenericComm`, injecting additional services before (and after) delegating transmission (and reception) responsibility to the internal component. The additional injected services focus on reducing link quality variation.

`ReliableComm` provides three salient features to improve network performance. First, the implementation introduces transmission queueing. The queueing strategy improves the utilization of the wireless radio without introducing unnecessary blocking delays in the calling context. Second, `ReliableComm` implements an acknowledgement and retransmission protocol to improve the *packet reception rate* (PRR) over each link. The implementation leverages MAC-level acknowledgement support, significantly reducing the additional traffic introduced by the acknowledgements. Third, `ReliableComm` implements a link quality monitor that estimates the PRR over each link based on empirical observation. These estimates are used to identify low-quality links over which messages are expected to be lost. This information is in turn used to discard messages destined for low-quality links. This not only improves the utilization of the radio (by avoiding transmissions that would be lost anyway), but also reduces overall network congestion, improving PRR *across* the network. While the individual services provided by `ReliableComm` are relatively straightforward, as we will see, the synergistic combination serves to provide significant benefits.

3 Testbed Overview

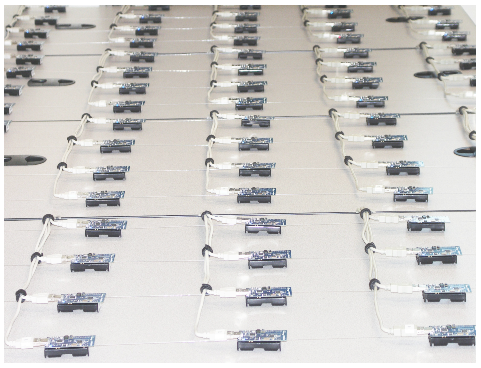

Figure 1: NESTbed Deployment

The evaluation of `ReliableComm` was performed on the Clemson University *NESTbed* system [3], a testbed infrastructure designed for evaluating large-scale sensor networks. The testbed is composed of 80 *Tmote Sky* [11] motes, arranged in a regular grid of 5 rows consisting of 16 motes each. Each matchbox-sized device includes an 8Mhz microcontroller, 48K of ROM, 10K of RAM, 1Mb of off-chip EEPROM storage, and a 2.4GHz IEEE 802.15.4 (*Zigbee*) wireless transceiver. This particular type of device is widely used in academic research, and is beginning to gain popularity in the commercial sector. A picture of the NESTbed deployment is shown in Figure 1.

As shown in the figure, each mote is attached to a centralized application and database server through a wired USB connection. Each connection provides power to the attached mote, and is used by the NESTbed server to program, control, and profile the device. The NESTbed server provides an extensive API for installing program images, inspecting source- and packet-level runtime data, and managing device power settings. The API is exposed through a remote interface used to perform the `ReliableComm` evaluation experiments presented here.

Due to (geographic) space limitations, the NESTbed deployment is far more dense than a typical network installation. To simulate greater geographic distribution, the API provides services for reducing the radio power level of each device. As we will see, by reducing transmission strength, the deployment admits of interesting wireless topologies, including multi-hop topologies with as many as five hops.

4 Performance Evaluation

We now turn our attention to the experiments performed to evaluate the link quality improvements provided by `ReliableComm`. The discussion is divided into two subsections, corresponding respectively to the two representative routing scenarios identified previously: (*i*) neighbor-to-neighbor and (*ii*) distributed convergecast. In each case, performance results collected using `GenericComm` are used as reference points.

4.1 Neighbor-to-Neighbor

The first evaluation scenario investigates the relative impact of `ReliableComm` on the link quality of *1-hop neighbors*. The experimental setup required the development of a custom application installed on the network using two deployment configurations. The first configuration used `GenericComm` for network communication; the second used `ReliableComm`. When installed, the application instructs the host device to transmit packets to each of its neighbors (in sequence) at a specified rate for a specified duration. The application includes a distributed time synchronization layer to ensure that each device begins and terminates at the same time[1]. Each mote records the number of messages sent on each link, as well as the number of messages received on each link. Dividing the number of packets received by the number of packets sent yields the PRR for a link. By comparing the results under the two deployment configurations, it is possible to compare the relative performance of `GenericComm` and `ReliableComm`.

In the first experiment, each mote was instructed to transmit at a rate of 4 messages per second for a duration of 20 minutes. To simulate geographic distribution, the radio power level of each device was reduced to 2 (on a scale from 1–31 [-25dBm–0dBm]). The link quality results for a small portion of the network (*i.e.*, the first row of 16 motes) are illustrated in Figure 2. The grid on the left corresponds to `GenericComm`; the grid on the right corresponds to `ReliableComm`. Each grid column represents a transmitting node; each row represents a receiving node. The cell shading at their intersection represents the PRR of the link defined by the transmitter-receiver pair, with black denoting

[1]The margin of error in the reported results is less than 1%.

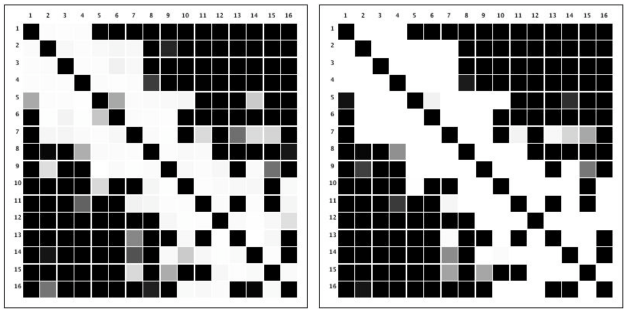

Figure 2: Link Quality Comparison (Power=2)

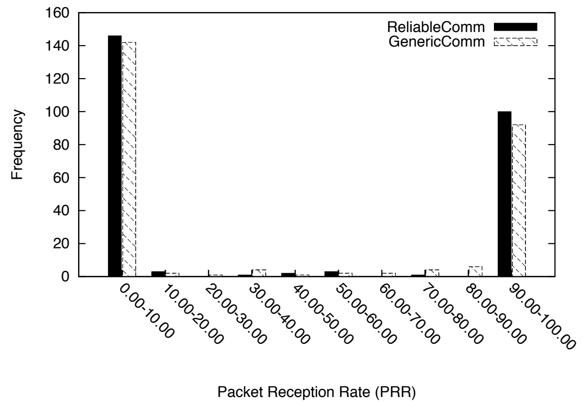

Figure 3: Link Quality Histogram (Power=2)

a PRR of 0% and white denoting a PRR of 100%. The cells are shaded on a linear scale between the PRR endpoints.

The figure shows that for these 16 motes, `ReliableComm` decreases link quality variability, generally separating links into two classes: dead links and high-quality links. Dark gray cells in the first grid tend to be black in the second. This shows that `ReliableComm` successfully identifies and avoids low-quality links. Light gray cells in the first grid tend to be white in the second. This shows that `ReliableComm` successfully improves the reliability of mid-quality links. White cells in the first grid remain white in the second, showing that `ReliableComm` has no impact on high-quality links.

The aggregate impact of `ReliableComm` on the first row of motes is summarized by the histogram shown in Figure 3. Ten link categories are considered, ranging from links of the lowest quality ($0\% \leq \text{PRR} < 10\%$) to links of the highest quality ($90\% \leq \text{PRR} \leq 100\%$). The vertical bars represent the number of links in each category. Although the benefits appear incremental when considering this small network subset, our performance claims are supported.

More striking are the results detailing the impact of `ReliableComm` on the network as a whole. The complete results of the 80 node experiment are shown in Figure 4. In this figure, the reduction in link quality variation provided by `ReliableComm` is clearly visible. The light gray bands running diagonally across the first grid correspond to mid-to-high-quality link regions within the network. The dark gray bands correspond to regions of low link quality. In the second figure, the light gray bands are almost entirely white; the dark gray bands are almost entirely black. This aggregate effect is summarized by the histogram shown in Figure 5. In effect, `ReliableComm` shifts mid-quality links to the highest link category, avoiding the lowest-quality links. The end result is improved network performance resulting from a three-fold increase in the number of high-quality links, and potential energy savings resulting from the avoidance of low-quality links.

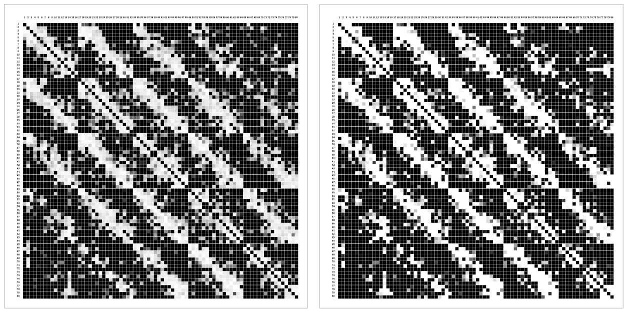

Figure 4: Link Quality Comparison (Power=2)

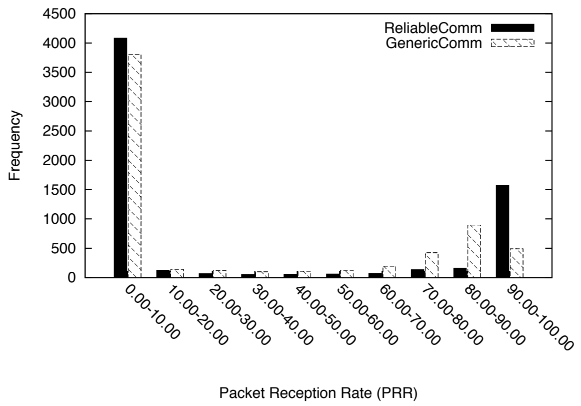

Figure 5: Link Quality Histogram (Power=2)

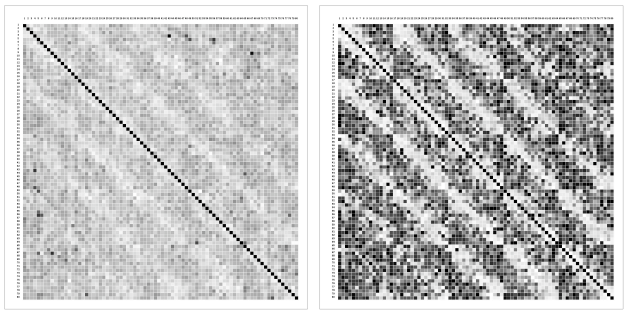

Figure 6: Link Quality Comparison (Power=4)

The second experiment was designed to investigate the performance of `ReliableComm` in dense deployment scenarios. To simulate reduced geographic distribution (as

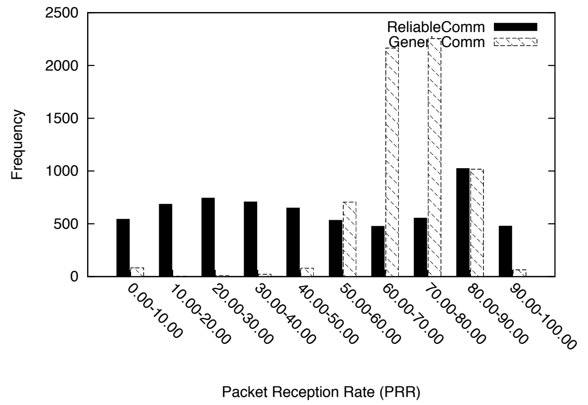

Figure 7: Link Quality Histogram (Power=4)

compared to the first experiment), the radio power level of each device was increased to 4 (from 2). Aside from this change, the experimental setup was unmodified.

The results of the second experiment are detailed in Figure 6, and summarized in Figure 7. Unfortunately, the results are surprisingly poor. The grid corresponding to `GenericComm` shows that there are relatively few low- or high-quality network links in the deployment. When `ReliableComm` is installed, the number of high-quality links increases (as evidenced by the increase in white cells), but the increase in low-quality links far outweighs this increase (as evidenced by the large increase in dark cells). As a result, in this deployment context, `ReliableComm` significantly degrades the performance of the network as a whole.

In view of the component's implementation, a careful analysis of the performance data reveals a limitation: In the dense deployment scenario, every mote is within the same 1-hop neighborhood. Further, every mote is transmitting at a rate of 4 messages per second. As a result, network congestion is the critical limiting factor in achieving high link quality. When `ReliableComm` is used, the retransmissions it introduces further degrade the quality of *most* network links. Indeed, the quality reduction is significant enough in many cases that mid-quality links under `GenericComm` are reduced to low-quality links that are periodically marked as *dead* by the `ReliableComm` link monitor. This accounts for the increase in the number of links in the lowest quality category. We will return to this limitation with a proposed solution in Section 6.

4.2 Distributed Convergecast

The second evaluation context investigates the end-to-end reliability improvements provided by `ReliableComm` in *multi-hop* scenarios. The focus is on distributed convergecast routing, generally used to route messages from various points in a multi-hop network to a well-known base station. Again, the experimental setup required the development of a custom application deployed using both `GenericComm` and `ReliableComm`. When installed, the application initiates a variant of the *TinyOS Beaconing* protocol [9] to construct a minimal spanning tree over the 80 node network. When the tree has stabilized, each node transmits messages to its parent at a specified rate for a specified duration. Each message is tagged with the sending node's distance from the root (measured in hops). The root node serves as an aggregation point, recording the number of messages received from each network tier. As in the first evaluation context, the application includes time synchronization support to ensure count accuracy[2].

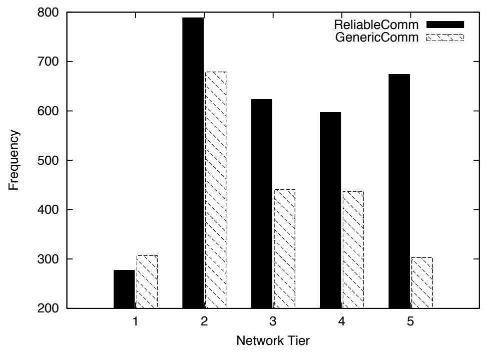

Figure 8: Packet Reception Count (Power=1)

When the experiment was executed, each mote was instructed to transmit at a rate of 4 messages per minute for a duration of 10 minutes. The radio power level of each device was configured to the lowest setting to simulate a sparse distributed network. The results of the experiment are summarized by the histogram shown in Figure 8. Each vertical bar represents the number of messages received by the root node from a particular network tier. As shown in the figure, `ReliableComm` provides significant end-to-end reliability improvements for tiers 2 through 5. There is, however, a modest reliability reduction for nodes in the first tier. The reason for this reduction is unclear. One possibility is based on the observation that tier-1 nodes forward more packets than lower-tier nodes, resulting in higher radio utilization. The additional forwarding introduced by `ReliableComm` could be causing a relatively higher percentage of messages *originating* from tier-1 to be lost. We plan to investigate this possibility as part of future work.

Additional studies were performed to investigate dense deployment scenarios. The convergecast experiment was repeated at power levels 2, 3, and 4. `ReliableComm` significantly outperformed `GenericComm` at power level 2, and performed equally well at power levels 3 and 4. This suggests that `ReliableComm` is suitable for a large class of deployment scenarios, especially those that are sparse and/or geographically distributed at large scales.

[2]The margin of error in the reported results is 0%.

5 Related Work

We are not the first to recognize the importance of link quality in determining application correctness and performance. Indeed, the literature is rich in this area. Due to space limitations, we consider only four of the most relevant elements of related work.

Kotz *et al.* [10] focus on analyzing radio stack models and implementations in the context of wireless networks. They address a range of issues, including link quality measurement, connectivity assessment in lossy environments, and transitional region analysis in low-power networks. The authors focus on analysis activities, providing suggestions for improving the design of experimental studies. Implementation techniques for improving link quality and avoiding low-quality links are not considered.

Chakeres *et al.* [1] provide an implementation of the *Ad Hoc On-Demand Distance Vector* (AODV) routing protocol to examine the effectiveness of `hello` messages in monitoring link status in mobile networks. They show that a number of factors influence tool-based monitoring accuracy. Similarly, Du *et al.* [5] present the *Neighborhood Link Quality Service* for wireless networks (NLQS). NLQS takes into account link asymmetry and timeliness concerns. The implementation also distinguishes between inbound and outbound neighbors, better accommodating the effects of link asymmetry on link quality estimates. In contrast to our work, neither of these authors consider the question of how to improve link quality, nor of how to avoid transmissions over low-quality links.

In Zhou *et al.* [13], the authors present a radio model to bridge the discrepancy between spherical models (typically used in simulation), and the physical reality of wireless signal propagation. They present a detailed study of the result of radio irregularity on both MAC and routing protocols, and suggest solutions. By contrast, our work tackles a two-fold problem by improving the reliability of mid-to-high-quality links, and by identifying (and avoiding transmissions over) low-quality links. This approach reduces link quality variation by way of an additional radio stack layer for TinyOS.

6 Conclusion

The goal of our work is to ensure predictable system performance by reducing the degree of link quality variation common to wireless embedded systems — with a focus on wireless *sensor* platforms. In support of this goal, we presented an alternative radio stack implementation for TinyOS [9] that improves the reliability of mid- to high-quality links, and identifies (and avoids transmissions over) low-quality links. We evaluated the performance of the implementation in two representative routing contexts using a fixed network deployment consisting of 80 *Tmote Sky* [11] nodes. Our performance results show that the implementation provides significant reliability improvements for a large class of systems, effectively reducing link quality variation to two categories (*i.e.*, dead links and high-quality links). The results also show that the implementation is ill-suited to certain deployments. In particular, the results suggest that the implementation is ill-suited to highly congested networks, where retransmissions can further degrade link quality. As part of our future work, we plan to investigate ways to remedy this limitation. One approach is to incorporate a network congestion monitor that informs the packet retransmission strategy.

Acknowledgements. This work was funded in part by a grant from the *National Science Foundation* (CNS-0520222), and a grant from the *South Carolina Space Grant Consortium*.

References

[1] I.D. Chakeres and E.M. Belding Royer. The utility of hello messages for determining link connectivity. In *The* 5^{th} *International Symposium on Wireless Personal Multimedia Communications*, volume 2, pages 504–508. IEEE Computer Society Press, October 2002.

[2] K. Chintalapudi et al. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2):26–34, 2006.

[3] A.R. Dalton and J.O. Hallstrom. An interactive, server-centric testbed for wireless sensor systems. Technical Report CU-DSRG-08-06-01, Clemson University (Dependable Systems Research Group), 2006.

[4] D.M. Doolin and N. Sitar. Wireless sensors for wildfire monitoring. In *SPIE Symposium on Smart Structures and Materials / NDE 2005*, pages 477–484. SPIE Press, March 2005.

[5] J. Du et al. Asymmetry-aware link quality services in wireless sensor networks. In *The 2005 IFIP International Conference on Embedded and Ubiquitous Computing*, pages 745–754. Springer, December 2005.

[6] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[7] D. Gay et al. Software design patterns for TinyOS. In *The 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 40–49. ACM Press, June 2005.

[8] C. Hartung et al. FireWxNet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *The* 4^{th} *International Conference on Mobile Systems, Applications, and Services*. ACM Press, June 2006. (to appear).

[9] J. Hill et al. System architecture directions for networked sensors. In *The* 9^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems*, volume 34, pages 93–104. ACM Press, November 2000.

[10] D. Kotz et al. The mistaken axioms of wireless-network research. Technical report, Dartmouth College (Computer Science), 2003.

[11] Moteiv Corporation. Tmote Sky datasheet. http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf, 2006.

[12] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[13] G. Zhou et al. Impact of radio irregularity on wireless sensor networks. In *The* 2^{nd} *International Conference on Mobile Systems, Applications, and Services*, pages 125–138. ACM Press, June 2004.