End-User Diagnosis of Communication Paths in Sensor Network Systems

Qing Cao, Dong Wang, Tarek Abdelzaher
Department of Computer Science, University of Illinois at Urbana-Champaign, IL, 61801
Email:{qcao2@cs.uiuc.edu, dwang24@cs.uiuc.edu, zaher@cs.uiuc.edu}

***Abstract*—In this paper, we present LiteView, an experimental toolkit for end-user diagnosis of communication path problems in sensor network systems. This toolkit provides an on-site, interactive environment that helps answering questions related to the instantaneous communication environments. For example, it allows users to identify broken links or asymmetric links, which are likely to become traffic bottlenecks. It also allows users to identify traffic hotspots by collecting round-trip delays of arbitrary pairs of nodes. Compared to the previous work on sensor network troubleshooting, this toolkit has two key differences. First, its utilities are generic and application-independent, allowing users to easily transfer experiences in managing one deployment to another. Second, its utilities are interactive, allowing users to quickly obtain the current status, optimize the deployment decisions, and observe their immediate effects. The utilities provided in this paper are thus promising to identify and solve communication path problems.**

I. Introduction

As sensor networks become more sophisticated, their successful deployment becomes an increasingly difficult task, abundant with pitfalls and traps. Consequently, planning, conducting, and managing real-world sensor network deployments efficiently is crucial for the success of both research efforts and commercial products. However, it is usually hard to conduct deployments strictly according to plans, as physical environments introduce a high degree of uncertainty, including unpredictable radio channel behavior, node positioning, and hard-to-replicate properties of the ambient environment. Uncertainty can also be introduced by the interactions of software modules on the same node or across nodes, which adds complexities to system dynamics. Consequently, it is necessary to empower developers with management utilities that allow them to interactively control nodes in the deployment phase. With the growing number and diversity of applications developed and deployed, such utilities should be designed independently of applications, and get standardized, so that users can accumulate experiences in the management of sensor network applications, and transfer such knowledge across deployments more easily.

In this paper, we are particularly interested in the communication aspect of management tasks, as our previous experiences show that communication layer failures contribute to a large category of malfunctions in deployed applications. We observe that if end-users could have accurate knowledge of the instantaneous connectivity between nodes in wireless sensor networks, they could improve the performance of the system through a variety of methods, such as tuning the radio power and channels, adding or removing nodes, or adjusting the directions of antennas. Such adjustments are especially promising to improve system performance in the final phase of deployments.

However, for resource-constrained sensor nodes, such as those formed by Mica-series nodes, obtaining link connectivity information, such as the contents of neighbor tables or the instantaneous link quality, is conventionally supported by applications. The interface to obtain such information varies across applications, even when they are implemented on the same operating system. This situation has several disadvantages. First, for applications that do not report link information, it is almost impossible to perform diagnosis on link connectivity. This causes deployed systems to act like black boxes that sometimes exhibit behavior that is hard to predict. Second, even with applications that do report link information such as their neighborhood and link quality, it usually varies across applications on how to collect and use such information. Therefore, each application introduces new learning curves, preventing users from performing generic management and diagnosis. Finally, as these reporting services are part of applications, they introduce extra overhead in both memory footprint and radio messages.

In contrast to these previous approaches, in this paper, we present a suite of interactive management tools for live collection and diagnosis of link quality in sensor networks. The key motivating goal of LiteView is to facilitate the understanding of the instantaneous communication environments, so that users can optimize deployment strategies accordingly. When nodes are deployed, the positioning, antenna orientation, and enclosures could have a significant impact on system performance. In one of our earlier sensor network deployments called the EnviroMic [2], we observed that the distance between nodes and their antenna directions considerably affected the communication layer performance as measured by end-to-end delays and packet losses. Ideally, such unpredictable environments should be probed in real-time, so that the deployment strategies can be optimized much like the way network administrators configure router settings in complex Internet environments. LiteView supports utilities that were previously available only for high-end networked systems, such as those based on embedded Linux platforms. LiteView extends these services to platforms where they are needed most, the low-power resource-constrained Mica platforms with only a few kilobytes of memory. As the core services, in

LiteView, we implement the *ping* command and the *traceroute* command, both of which were re-designed for the low-power, 802.15.4-compliant wireless environments of Mote platforms, to help probing current network conditions. We also implement utilities to manage neighborhoods. With these facilities, users get visibility on the way of routing tree construction, the patterns of packet generation, and the hotspots of lost packets.

We systematically implemented LiteView on the LiteOS operating system [3]. We plan to release LiteView as part of the updates for future versions of LiteOS. The core services provided by LiteView are closely integrated with the utilities provided by LiteOS for wireless application deployments, such as the tools for trouble-shooting application behavior through interactive debuggers, and the support for understanding system dynamics based on on-demand logging of internal events. To the best of our knowledge, LiteView is the first application-independent toolkit that supports interactive path diagnosis for resource-constrained sensor networks. Our experiments demonstrate that such integration serves our needs well. Compared to the previous work, LiteView has two unique features. First, LiteView is application independent. In other words, the management capabilities of LiteView are not limited to any particular deployed applications. This allows LiteView to be extremely flexible for management purposes. Second, LiteView provides a very friendly user interface, by extending the interactive shell-like environment of LiteOS. We hope that these two features allow LiteView to be easily adopted by end-users.

The remainder of the paper is organized as follows. Section II reviews the related background for the development of LiteView. Section III describes the management solutions supported by LiteView. Section IV describes the implementation details. Section V presents the evaluation of LiteView. Section VI concludes the paper.

II. Background and Related Work

Management services for networks have been developed for a long period of time. The Simple Network Management Protocol (SNMP) [4], developed in the late 1980s by ISO, used to be the standard management protocol for conventional TCP/IP networks. However, the unique characteristics of wireless sensor network make SNMP not applicable. Ad Hoc Network Management Protocol (ANMP) [5] is an extended version of SNMP that enables SNMP to work for wireless networks. But it is not generic enough for low-end wireless sensor networks.

Recent efforts aimed for specific management services for sensor networks. The Management Architecture for Wireless Sensor Networks (MANNA) protocol [6] is a management solution specifically designed for WSNs. It provides a general framework for policy-based management of sensor networks by collecting dynamic management information from the MIBs (Management Information Bases), which it then maps to a wireless sensor network deployment. However, the MIB update, which is a centralized operation, is extremely energy consuming. Another service, the Sensor Network Management Protocol (sNMP) [7], is a management framework that defines sensor models representing the current state of sensor network and defines various network management functions. It provides algorithms and tools for retrieving network state information by executing network management functions. However, the MIB-based framework suffers from similar problems as MANNA. A third protocol, the Sensor Network Management System (SNMS) [8] as proposed by Tolle and Culler, is an interactive system for monitoring the health of sensor networks, which provides two core management services: query-based network health data collection and active event logging. The query service allows users to monitor physical parameters of the sensor network environments, while the event-driven logging service allows sensor nodes to report their data once they have met the specifications set by users. L-SNMS [9] improves SNMS by incorporating an RPC mechanism inspired by an interactive debugging tool for WSN development, Marionette [10]. The RPC mechanism offers users with necessary support to access the functions and variables of applications running on the sensor nodes during runtime. These efforts are significantly different from ours in that they are based on the monolithic kernel of TinyOS. Therefore, they only allow users to modify variable state without providing more comprehensive controls at the process level or application level.

Complementary to the management tasks is the debugging of wireless sensor networks in the deployment phase. As simulation based debugging proved to be limited in its ability to simulate real-time network dynamics, recent efforts have aimed to provide deployment stage visibility for better understanding of software behavior. Visualization tools are representative of earlier efforts. The Emstar [11] visualizer displays both link quality and neighbor connectivity. However, such services require full knowledge of applications, as well as cooperation from software developers. More recently, some debugging services implemented diagnosis information reports as modules of applications. For example, MintRoute [12] includes periodic transmission of neighbor tables to help debugging at the sink. However, it neither supports user-defined metrics after deployment, nor performs failure analysis. It also introduces overhead no matter whether such information is of interest. Sympathy [13] combines several techniques mentioned above and focuses on the collection of various types of metrics on sensor nodes to detect and identify failures in the network. The collected data are routed by the sensor network itself back to the sink where the evaluation is carried out. Sympathy's failure detection and localization are based on the assumption that a continuous data flow exists in the network. It uses the collected metrics to localize failures following a decision tree based on previous experience and heuristics. However, Sympathy is implemented as a fixed module, and is not user interactive.

III. Management Solutions

A. Design Goals

The major goal of LiteView is to enable interactive in-situ management of deployed sensor network applications. Its design goals include:

- **Effectiveness**: This metric is measured by the ability of LiteView to collect real-time link quality as well as the round-trip delays with acceptable response time.
- **Efficiency**: In sensor platforms, resource constraints (on both CPU and memory) are significant, which makes it critical to use resources efficiently. This efficiency is measured by the footprint of LiteView and its communication overhead. In our implementation, we follow a highly modular design, where the implemented commands will introduce *zero* extra overhead if not activated.
- **Simplicity**: Aiming to simplify the management of sensor network applications, LiteView must be easy to use. In other words, simple user interfaces are preferred. Experiences tell us that simple tools tend to gain more popularity and persist longer while more complex tools sometimes run into usability barriers and become less desirable. Our design achieves this goal by integrating the commands supported by the shell of LiteOS, making them extremely easy to use.
- **Extensibility**: The extensibility of LiteView is measured by its modularity, where it can work with new communication protocols without recompilations. Our implementation achieves this goal by designing and implementing a highly compact, modular communication stack where multiple routing protocols can co-exist, and there is no redundancy between protocols.

The above goals provide guidelines throughout the design and implementation of LiteView.

B. Management Solutions

As sensor networks are formed by low-power networked embedded nodes, one of the most tricky problems for successful deployments is their communication layer behavior. Common management problems include understanding the radio behavior in realistic environments (e.g., what is the radio range and the link quality?), channel selection and management (what is the current channel?), neighborhood management (has the current node lost connection with all other nodes?), and path diagnosis (where is the major delays for a particular path?). However, there have been no comprehensive studies on how such problems can be addressed by a generic framework of utilities. LiteView aims to answer these questions with a suite of commands, organized into radio configurations, neighborhood management, link estimation, and path diagnosis.

1) Radio Configurations: For our target platform, MicaZ nodes, we provide commands for adjusting both the transmission power and the channel assignments. The following two types of commands are supported. First, the adjustment of radio transmission power. The CC2420 radio installed on MicaZ motes supports programmed output power ranging from -25dBm to 0dBm. LiteView allows users not only to view the current power level, but also to adjust it to desired levels. Second, the assignment of radio channels. For instance, the CC2420 radio chip of our target hardware platform, the MicaZ nodes, supports 16 channels. Similarly, users are allowed to view the current channel or adjust it as desired.

2) Neighborhood Management: The second group of operations is neighborhood management. In our design, we modified LiteOS so that the kernel maintains a list of neighbors for each node, including their node names, identifiers, and link quality. The motivation for this design decision is as follows. As multiple communication protocols are likely to rely on neighborhood information, it is not cost-effective to allow each protocol to maintain an independent version of neighbor tables. Instead, it is more efficient to provide neighborhood management as part of kernel services, which both users and applications can access via system calls.

The role of LiteView lies in that it exposes the neighbor table for management purposes. More specifically, it allows users to read the neighbor table of the current node (with or without link information), blacklist a neighbor (temporarily preventing communicating with this neighbor), or remove a neighbor from a blacklist. The blacklist mechanism temporarily modifies the behavior of communication protocols when they construct routing and transport structures.

3) Link Profiling: It is often desirable to know link quality, such as the loss ratio and the end-to-end delay, of particular pairs of nodes. LiteView provides one familiar command, *ping*, for actively probing other nodes and collecting link quality information. This command is invoked using the identifer of another node as its parameter. Other optional parameters include the number of rounds of packets, and the length of probe payload. When this command is invoked, the sender will send a probe packet to the destination node, which, in turn, sends a response packet back. This process may be repeated for multiple rounds and/or with different packet lengths.

For example, in our testbed, we assign names following IP conventions to each node as their names. We then use the *ping* command after logging into the node **192.168.0.1** to estimate the link quality between itself and another node **192.168.0.2**. The sample output of the *ping* command is shown as follows (slightly reformatted for printing considerations). Here, we are operating under the LiteOS shell that provides a Unix-like environment. The target platform is a testbed composed of thirty MicaZ nodes.

```
$pwd
    /sn01/192.168.0.1
$ping 192.168.0.2 round=1 length=32

    Pinging 192.168.0.2 with 1 packets
     with 32 bytes:
    RTT = 4.7 ms, LQI = 108/106,
    RSSI = -1/8, Queue = 0/0
    Power = 31, Channel = 17
```

```
Ping statistics:
    Packets = 1
    Received = 1
    Lost = 0,
```

Note that we report both LQI and RSSI readings. More specifically, the LQI metric is a characterization of link quality based on received packets, and is defined in the 802.15.4 standard [1]. In CC2420, LQI is implemented based on the average correlation value of each first 8 symbols following the packet SFD (Start of Frame Delimiter). A correlation of around 110 indicates the highest quality while a value of 50 the lowest. The RSSI, on the other hand, stands for the Received Signal Strength Indicator, and is based on averaged readings of eight symbol periods of 128 μs. It has a linear relationship with the power level of received packets. For example, a RSSI reading of -20 indicates that a RF power of approximately -65dBm. RSSI is different from LQI in that it is more related to the signal strength, while LQI could also be affected the presence of radio interferers.

Finally, note that the *ping* command in the previous example is only for one-hop purposes. A multi-hop *ping* command is also available for profiling paths, and will be described in the next section.

4) Path Profiling: In addition to profiling the quality of individual links, LiteView also allows profiling link quality of multiple hops, or paths. We implement two commands for this purpose: the multi-hop *ping* command, and the specifically designed *traceroute* command. Both commands deliver probe packets over multiple hops using underlying routing protocols, and collect path link quality information as packets travel multiple hops in the network.

The multi-hop *ping* command is implemented as follows. Its probe packets are forwarded just like any other data packets, except that it collects link quality information along the path. When this packet is received by the destination node, a reply is sent back, where the previously collected link quality information is inserted into the reply packet. On its way back, the packet collects additional link quality information, until the reply packet is received by the sender, which prints out the end-to-end round trip time and the link quality.

On the other hand, the *traceroute* command is implemented differently. It operates on a per-hop basis, where a reply packet will be generated on each hop along the path. This design choice allows round-trip time of each hop to be measured and sent back to the source node. Because of this design, the *traceroute* command is fundamentally more scalable compared to the multi-hop *ping* command. Therefore, the *traceroute* command supports collecting additional information, such as the neighbor table contents of each node along the path. The functionality and implementation of the multi-hop *ping* and *traceroute* will be described in the next section.

Following is a typical output of the *traceroute* command. First, note that the RTT values reported here are for individual hops rather than for end-to-end paths. Second, note that we use the parameter *port* to specify the routing protocol used to deliver packets over multiple hops. In this example, we let the geographic forwarding protocol listen on the port number 10, so that it will be selected by the *traceroute* command.

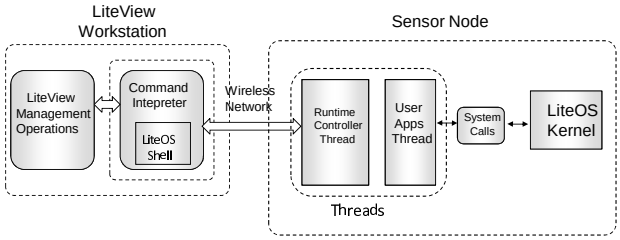

Fig. 1. System Architecture

```
$pwd
    /sn01/192.168.0.1
$traceroute 192.168.0.3
 round=1 length=32 port=10

    Reaching 192.168.0.3 with 1
    packets with 32 bytes:
    Name of protocol: geographic forwarding

    Reply from 192.168.0.2
        RTT = 4.9 ms, LQI = 106/107,
        RSSI = 1/2, Queue = 0/0
    Reply from 192.168.0.3
        RTT = 4.7ms, LQI = 105/103,
        RSSI = -1/0, Queue = 0/0

    Traceroute statistics:
        Packets = 1,
        Received = 1,
        Lost = 0,
```

IV. Implementation

We implement LiteView on the MicaZ hardware platform. Each MicaZ node has an Atmega128 microcontroller, 4KB RAM, and 128K programmable flash. The implementation can also support the IRIS platform with moderate changes. LiteView is implemented on LiteOS 1.0, an operating system developed for the aforementioned hardware platforms.

A. Implementation Challenges

While the interfaces of the commands, such as the neighborhood management, the *ping* and *traceroute* commands, are sufficiently simple, their implementation has several unique challenges in the resource-constrained environment of sensor nodes. This section lists three key challenges that we addressed in our implementation.

1) Protocol Independence: The first challenge is that the implemented commands need to be protocol independent, i.e., they should allow users to switch between different routing layer or MAC layer protocols. This design decision allows users to compare between protocols. For example, it

is common that because protocols follow their own ways to construct their internal routing structure, they exhibit different performance. To select the optimal combination of protocols, users may install each protocol sequentially, and measure the system performance. Therefore, it is desired that the *ping* and *traceroute* commands should support multiple protocols without the need for re-compilation. This design choice also indicates that the routing layer protocols should not contain any functionality specifically designed for the management tools. Instead, the implementation should guarantee complete isolation between the command module and the protocol module.

2) Passing Parameters: The second challenge is related to command parameters. Specifically, the implementation should facilitate a mechanism where users can provide runtime parameters to the command executables, such that their behavior can be decided at runtime rather than hard-coded. For example, the *ping* command should be able to accept the address of the destination node as its parameter.

3) Packet Manipulation: The third challenge is related to collecting path information, such as the LQI and RSSI readings, both of which are closely related to link quality. The key challenge is that to prevent corrupting contents of data packets, we should not directly store link quality information into the original payload of packets. An ideal design, instead, should provide appropriate packet manipulation mechanisms without modifying the original packet payloads.

B. System Architecture

The overall system architecture of LiteView is illustrated in Figure 1. LiteView consists of a command interpreter on the client side, and a runtime controller on the node side. From the perspective of the client, the LiteView command interpreter extends the LiteOS shell. From the perspective of applications, the LiteView runtime controller is a co-existing process that will interact with other processes. The command interpreter and the runtime controller are connected via a common communication middle layer, where commands are translated into broadcasted messages that are received by the runtime controller.

Our design goal of simplicity calls for a simple interface for users to interactively apply management tasks. The user interface provided by LiteView is an extension of the interactive shell of the LiteOS operating system. Compared to the latter, it supports additional operations and management tasks.

The command interpreter carries out three tasks. First, it translates each user command into a sequence of radio messages. Each message header corresponds to one unique type, while the command parameters are embedded into message bodies. Second, it keeps track of the context of user management operations, such as the current directory that users are located at, so that the command interpreter can answer certain user queries without the need for contacting remote nodes. Finally, the command interpreter communicates with the runtime controller running on the nodes following a reliable one-hop communication protocol that exchanges both commands and replies.

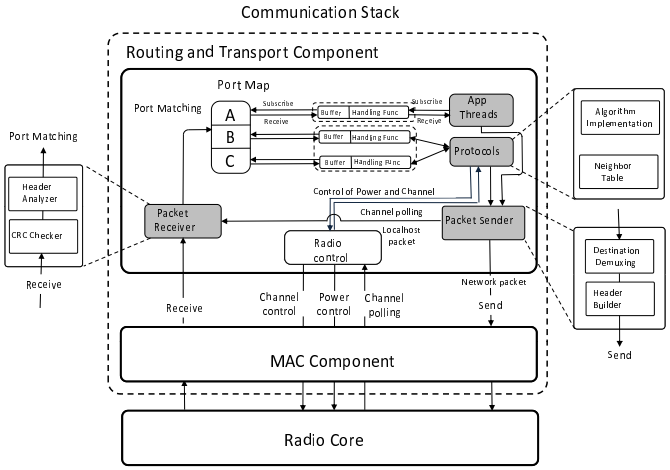

Fig. 2. Communication Stack Architecture

The reliable message exchange protocol works as follows. For commands interpreted into one single packet, one acknowledgement packet, combined with a timeout mechanism, is sufficient. For commands translated into a sequence of packets, the protocol operates in batches, with one acknowledgement packet for each batch. The number of packets in each batch is dynamically adjusted based on link quality: a smaller batch size is preferred when packets are more likely to get lost. The lost packets are detected at the node side by detecting missing sequence numbers. Finally, if the management workstation is operating on a group of nodes, these nodes wait for random backoff delays before sending responses, so that their packets will not collide. Our practical measurements show that this simple protocol works reliably well for one-hop communication.

On the node side, LiteView implements a runtime controller that interacts with the command interpreter. This controller serves two purposes. First, it provides comprehensive visibility on neighborhood management by allowing users to view state of neighbors kept by the kernel. This information is collected either by invoking the APIs provided by the underlying OS, or by directly reading memory addresses. Second, it executes user commands, where it interacts with communication protocols to send or receive messages. Unlike other built-in commands supported by LiteOS, the commands supported by LiteView are executed as individual processes.

C. Building Blocks

The LiteView runtime controller comprises of several components, which we refer to as *building blocks*. In this section, we describe how these building blocks are implemented.

1) The Communication Infrastructure: Both the *ping* command and the *traceroute* command are implemented based on a port-based communication stack. Therefore, before describing the details of both commands, we describe our implementation of the communication stack.

Figure 2 shows the architecture in this communication stack. Observe that both the receiving (on the left) and the sending (on the right) operations are illustrated. When the sender intends to deliver packets, it puts the destination address as well as the port number for the destination node into the packet header. The packet is then delivered to the MAC component and broadcasted over the radio. When the packet is received by a neighbor, its CRC field is first checked for integrity. If this packet is intended for the current node, its port number is matched against each process that is listening to incoming packets. The thread that has a match in port number is considered the right thread for the incoming packet. The contents of the packet are then copied into the internal buffer of the thread, which is in turn waken up to handle the incoming packet. For example, this listening thread could be the routing protocol that will continue to forward the packet along the path.

The subscription-based communication infrastructure has the following advantages. First, it achieves complete isolation between the protocol implementation and the applications. As the runtime controller for LiteView is a thread by itself, this design allows multiple protocols to work with the same set of LiteView commands. Second, this design significantly reduces the system complexities, as the system now exhibits a clearly layered architecture, where the only shared data between layers are packets themselves. Therefore, a developer for applications does not need to worry about the implementation details of the underlying communication protocols.

2) Neighborhood Management: LiteView provides visibility on the up-to-date information of the neighborhood of a particular node, including the number of neighbors, the links connecting the current node with its neighbors, and operations related to blacklists. The mechanisms for implementing these commands are simple: to obtain the neighbor information, the runtime controller directly accesses the memory contents holding neighbor tables in the kernel. For operations related to blacklists, the kernel associates a field to each neighbor entry that specifies whether or not the current neighbor is considered *enabled*. By modifying the state of this field, LiteView allows users to temporarily add or remove neighbors into or from a blacklist.

From the perspective of users, the neighborhood management proceeds as follows. The user first enters the neighborhood management mode by invoking the *neighborsetup* command. The user may then choose from several commands for neighbor table management. For example, the *list* command allows users to view the contents of the neighbor table. The *blacklist* command allows users to add or remove nodes to or from blacklists (by using different parameters). The *update* command allows users to configure neighborhood settings, such as the frequency of neighbor beacon exchanges. By default, all commands have a response delay of 500 milliseconds.

3) Collection of Path Quality Metrics: A central problem of the LiteView architecture is how to collect path quality metrics, as measured by LQI and RSSI values. Note that such metrics can only be obtained at the receiver side after the packet has been successfully received and decoded. To collect path quality metrics without modifying data payloads, we implement a standardized mechanism in routing protocols called *link quality padding*, which can be enabled or disabled according to application needs. The padding works as follows. Normally, in the routing layer, we keep a default payload of 64 bytes, serving as the upper limit on the length of data payloads. If the actual length of data payloads is shorter, say, 32 bytes, only the actual data payload is transmitted over the air. When link quality padding is enabled, the routing layer utilizes the extra bytes that are normally not transmitted over the air for storing link quality metrics, including LQI and RSSI values. For example, the routing layer will pad two metrics, LQI and RSSI, to the end of the data payloads at each hop. Note that the packet will be longer and longer when it is delivered along the path. Therefore, one tradeoff of this design is that as the maximum space for padding is limited, so is the maximum number of hops packets can travel with padding enabled. When this service is used for implementing the multi-hop *ping* command, as the probe packet has a payload of 16 bytes, as each hop takes two bytes in padding, a packet could at most travel 24 hops before the padding runs out of space. This is sufficient for most applications. The *traceroute* command, on the other hand, does not require padding as it will send back link quality information of each hop along the path. Therefore, it is more scalable compared to the *ping* command.

4) Passing Parameters: One key mechanism for flexible management commands is the support for runtime parameters. For example, for the *ping* command, we typically supply the destination address as its parameter. As the LiteOS operating system does not provide a mechanism for passing parameters to executables, we implement a new system call for passing parameters. More specifically, we keep the parameters, such as the identifiers of destination nodes, into a buffer kept by the runtime controller. The address of the buffer is accessible by a system call function provided by the kernel. When a new process is started, it invokes the system call to obtain the address of the parameter buffer. If no parameter is supplied, the buffer will start with a "\0". Otherwise, the process will find the parameter string in this buffer. Multiple parameters could be separately by spaces, and it is up to the processes to parse them correctly.

5) The Ping Command Implementation: In this section, we describe the implementation of the *ping* command. Its compiled binary image consumes 2148 bytes of flash and 278 bytes of static RAM. This overhead is well acceptable even on the resource-constrained MicaZ nodes. This command is implemented as an individual thread running concurrently with the kernel, on both the sender side and the receiver side. It subscribes to a unique communication port, so that two *ping* instances can exchange packets via communication links. On the sender side, the process first gets the current timestamp using a high-resolution, cycle-accurate timer. This process then sends out a probe packet to the receiver. When this packet is received by the process running on the receiver

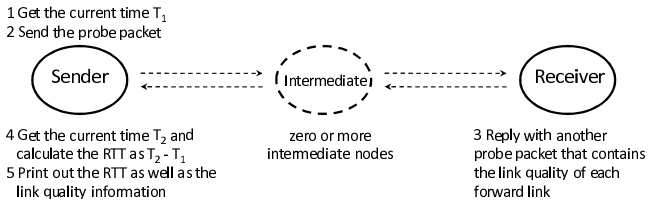

Fig. 3. The Mechanism of the Ping Command

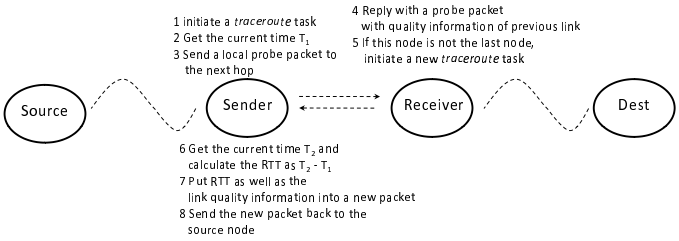

Fig. 4. The Mechanism of the Traceroute Command

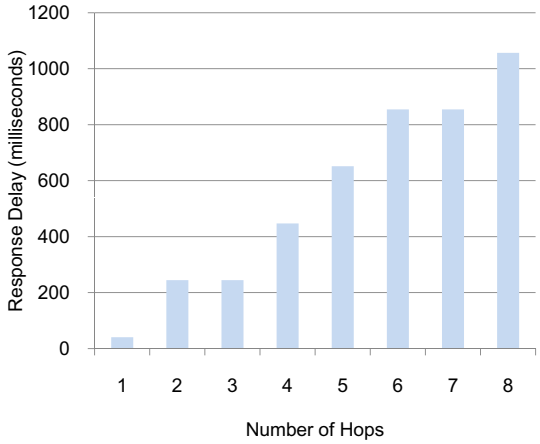

Fig. 5. Evaluation on the Response Delay of the Traceroute Command

node, the receiver sends out a reply packet, which contains the link quality information (LQI value and the RSSI value) of the incoming packet. Such information is only available after the packet is received. As the sender receives the reply, it calculates the difference in the timestamps as the RTT time for incoming packets. The whole implementation is shown in Figure 3. Note that in this implementation, we only obtain timing information on the same node (the sender). Therefore, no network level synchronization service is needed.

The *ping* command accepts several parameters, including the length of the probe packet and the destination node. Note that if the *ping* command is going to send a probe packet that travels multiple hops, an underlying routing protocol is needed. In this case, an additional parameter for specifying the port number is required. The probe packet is then delivered to the process that is listening on this port, and uses this protocol as the underlying carrying mechanism.

6) The Traceroute Command Implementation: The implementation of the *traceroute* command is more complicated compared to the *ping* command. Its compiled image consumes 2820 bytes of flash and 272 bytes of static RAM. When this command is invoked, the runtime controller first initializes the network by starting the traceroute process on each node along the path. Then, on each hop along the path, the intermediate node temporarily becomes a sender, and will initiate a traceroute task. In this task, the node will behave similarly to a sender in the *ping* command. More specifically, it sends a probe to the next node, if it is not a destination as specified by the original *traceroute* command. It then waits for the reply from the next node, and obtains both the RTT value and the link quality information. This intermediate node then puts such information into a report packet, and delivers it to the source node that started the *traceroute* command. Note that this packet contains the details on the link quality information for only one hop along the path. Once the source node receives the packet, it will print out the details on the hop information. For a path composed of multiple hops, the source receives multiple reports from different nodes, so that it gathers the path quality information of the entire path.

V. Evaluation

Having systematically implemented the LiteView toolkit on the MicaZ platform, we now describe its evaluation results. Our primary evaluation metrics include: response delays, path quality information with different levels of power settings, and the overhead of collecting path metrics.

A. Response Delays

The first metric we evaluate is the response delay for using the commands in LiteView. Both the neighborhood management and the single-hop *ping* command has a response delay of 500 milliseconds, which is consistent with most other commands in LiteOS. This period of time is intentionally longer than needed to receive the replies from the network, as we use extra waiting time to allow nodes to add random waiting time before sending back replies.

One notable exception to the 500 milliseconds response time is the *traceroute* command. In this case, the sender has to wait for all the nodes along the path to send back reports before it can finish the task. We measured the response delay with a testbed of eight hops in diameter. Figure 5 shows the response delay for receiving the packets from different hops in one typical experiment. Note that, while the delay typically increases, it is possible that some packets are received in almost the same time because the underlying routing protocol has a queueing mechanism to hold packets temporarily. If the routing layer determines that the channel is busy, it will add random jitters before sending out packets in the queue. Such backoff mechanisms may cause some packets along a path to be delivered back-to-back. Therefore, these packets arrive at the source node at almost the same time. This is the case with link 6 and 7. Here, packet 7 catches up with packet 6, and both are delivered back-to-back to the source node.

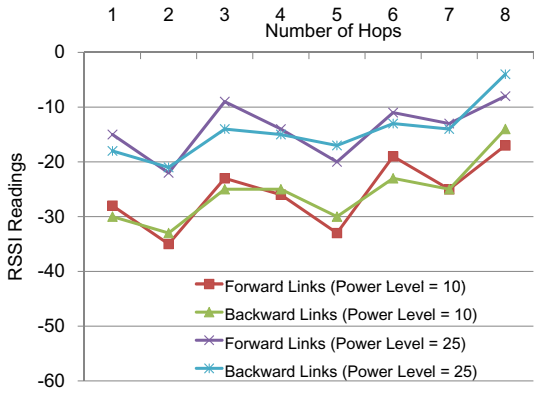

Fig. 6. Collected RSSI Values using Traceroute

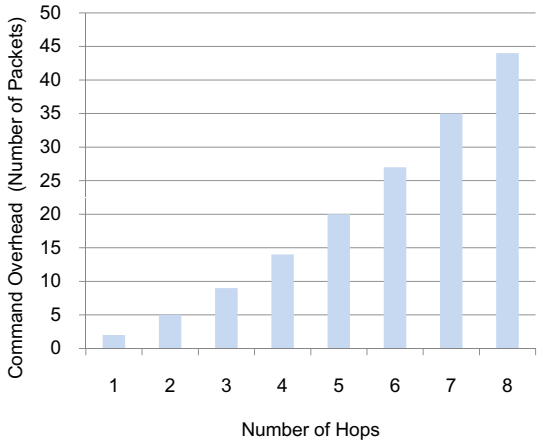

Fig. 7. Traceroute Command Overhead

B. Path Quality Information

The second metric we are interested in is the effectiveness of LiteView in collecting path quality metrics, such as the parameters of LQI and RSSI. Our experiments demonstrate the LiteView is very effective in obtaining such parameters, and we use RSSI as an example in this experiment. Figure 6 shows the collected RSSI values with two different power level settings, at 10 and 25, respectively. Note that the different power levels have a significant effects on the reported RSSI values. As the results are obtained within a few seconds, this figure demonstrates that with the use of the LiteView toolkit, it is very easy and convenient to obtain real-time radio statistics of the communication path in sensor networks.

C. System Overhead

The third metric we are interested in is the system overhead. We are mostly interested in the communication overhead of invoking the commands. For one hop protocols such as *ping*, the overhead is sufficiently small, usually only two packets. For multi-hop protocols, the overhead is more significant, but still very well acceptable. For example, Figure 7 shows the number of control messages as measured by invoking the traceroute command with different number of hops in diameter. Note that the overhead grows almost linearly, with fewer than 50 control packets for 8 hops. Therefore, we consider this to be very acceptable.

VI. Conclusions and Future Work

In this paper, we described LiteView, a suite of on-site management tools that provides interactive diagnosis of communication behavior. We have implemented LiteView based on the LiteOS operating system running on MicaZ nodes, and we will release the LiteView suite of tools in future LiteOS releases.

In the future, we intend to use LiteView to manage sensor network applications. Our preliminary user experiences are satisfactory: we can now quickly identify traffic hotspots and link quality at deployment phase, so that we can get a much better picture on the instantaneous network conditions. The response delays and overhead introduced by LiteView are both very acceptable even on very resource-constrained sensor nodes.

References

[1] IEEE std. 802.15.4 - 2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs) http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf

[2] Luo, L., Cao, Q., Huang, C., Abdelzaher, T., Stankovic, J.A., Ward, M.: Enviromic: Towards cooperative storage and retrieval in audio sensor networks. In: Proceedings of the 27th International Conference on Distributed Computing Systems. (2007)

[3] Cao, Q., Abdelzaher, T., Stankovic, J., He, T.: The LiteOS Operating System: Towards Unix-like Abstractions for Wireless Sensor Networks. In: Proceedings of the IPSN Conference. (2008)

[4] Kurose, J., Ross, K.: Computer Networking: A Top-Down Approach Featuring the Internet. Addison-Wesley Longman Publishing Co., Inc., Boston. (2002)

[5] Chen, W., Jain, N., Singh, S.: ANMP: Ad hoc network network management protocol. IEEE Journal on Selected Areas in Communications. (1999)

[6] Ruiz, L.B., Nogueira, J.M., Loureiro, A.: Manna: A management architecture for wireless sensor networks. IEEE Communications Magazine, 41(2):116-125. (2003)

[7] the sNMP Framework, http://www.research.rutgers.edu/~bdeb/sensor_networks.html

[8] Tolle, G., Culler D.: Design of an application-cooperativemanagement system for wireless sensor networks. In: Proceedings of the second European Workshop on Wireless Sensor Networks. (2005)

[9] Yuan, F., Song, W., Peterson, N., Peng, Y., Wang, L., Shirazi, B., LaHusen R.: A Lightweight Sensor Network Management System Design. In: Proceedings of the Sixth Annual IEEE International Conference on Pervasive Computing and Communications. (2008)

[10] Whitehouse, K., Tolle, G., Taneja, J., Sharp, C., Kim, S., Jeong, J., Hui, J., Dutta, P., Culler, D.: Marionette: Using rpc for interactive development and debugging of wireless embedded networks. In: Proceedings of the Fifth International Conference on Information Processing in Sensor Networks. (2006)

[11] Girod, L., Elson, J., Cerpa, A., Stathopoulos, T., Ramanathan, N., Estrin, D.: EmStar: A software environment for developing and deploying wireless sensor networks. In: Proceedings of USENIX. (2004)

[12] Girod, L., Stathopoulos, T., Ramanathan, N., Estrin, D.: Tools for deployment and simulation of heterogeneous sensor networks. Technical report. (2004)

[13] Ramanathan, N., Kohler, E., Girod, L., Estrin, D.: Sympathy: A debugging system for sensor networks. In: Porceedings of EmNets. (2004)

[14] Chen, B., Peterson, G., Mainland G., Welsh, M.: LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics. In: Proceedings of DCOSS (2008)