SRCP: Simple Remote Control for Perpetual High-power Sensor Networks

Navin Sharma, Jeremy Gummeson, David Irwin, and Prashant Shenoy

University of Massachusetts, Amherst
Department of Computer Science
{nksharma,gummeson,irwin,shenoy}@cs.umass.edu

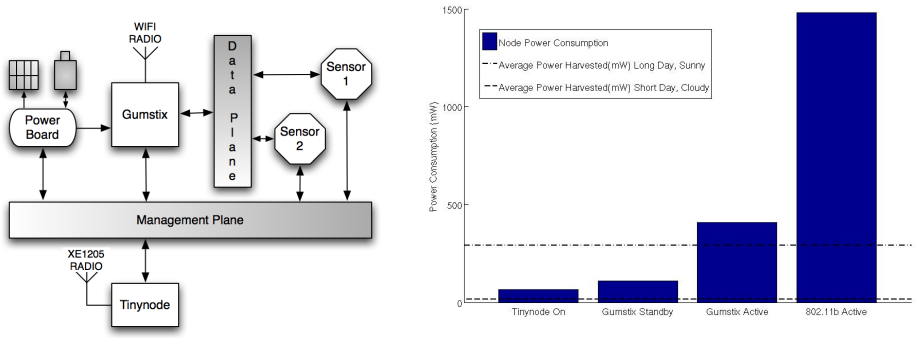
Abstract. Remote management is essential for wireless sensor networks (WSNs) designed to run perpetually using harvested energy. A natural division of function for managing WSNs is to employ both an in-band data plane to sense, store, process, and forward data, and an out-of-band management plane to remotely control each node and its sensors. This paper presents *SRCP*, a Simple Remote Control Protocol that forms the core of an out-of-band management plane for WSNs. SRCP is motivated by our target environment: a perpetual deployment of high-power, aggressively duty-cycled nodes capable of handling high-bandwidth sensor data from multiple sensors. The protocol runs on low-power always-on control processors using harvested energy, distills an essential set of primitives, and uses them to control a suite of existing management functions on more powerful main nodes. We demonstrate SRCP's utility by presenting a case study that (i) uses it to control a broad spectrum of management functions and (ii) quantifies its efficacy and performance.

1 Introduction

Perpetual wireless sensor networks (WSNs) consist of nodes that harvest environmental energy (*e.g.*, solar, wind, thermal, vibration) to indefinitely sustain data collection, storage, processing, and transmission without physical interaction [1]. Perpetual operation is ideal for WSNs composed of nodes that are time-consuming to construct and deploy. However, the long lifetime of perpetual WSNs elevates the importance of remote management functionality. Effective remote management has three characteristics:

– **Visibility.** A sensor node is visible if its state *is known or can be queried*. High visibility permits simple, direct monitoring of each node with high accuracy, while low visibility forces complex, indirect monitoring based on collected sensor data, past node states, or neighbor states. Visibility is essential

[1] This research was supported in part by an UMass President's Science and Technology award and NSF grants CNS-0626873, CNS-0615075, CNS-0520729, and EEC-0313747. We also acknowledge Deepak Ganesan for providing us feedback on early versions of this work.



(a) Management Plane Overview (b) Power Consumption

Fig. 1. Our prototype (a) uses a management plane. Power states are shown in (b) from a typical sunny summer day and cloudy winter day.

for discovering the software bugs or hardware faults that impair WSN operation; recent work proposes elevating low-cost visibility to a fundamental WSN design principle [21].

- **Accessibility.** A sensor node is accessible if its state is known or can be queried, and *can also be altered*. High accessibility permits simple, direct remote node maintenance, including altering application-level software, kernel-level software, and hardware states (*e.g.*, power states). Low accessibility increases the scope of problems that require physical access to a node.
- **Interactivity.** A sensor node is interactive if its state *is both visible and accessible with tolerable latency*. High interactivity permits a controller to react to visible changes in the WSN by accessing a node and changing its state, and quickens the upgrade-test-debug development cycle required to produce robust software. Low interactivity limits the WSN's capability to adapt its operation to unexpected operational or environmental changes, and slows or impairs the software development cycle.

Visibility, accessibility, and interactivity are highly correlated with a node's duty cycle, since a controller is unable to query or alter state while a node is powered down. As a result, simultaneously satisfying all three characteristics is challenging for resource-constrained WSNs. Previous approaches primarily address only a single aspect of remote management using *in-band* techniques that share a single wireless channel and node processor between both management-centric and data-centric tasks [11, 18, 20, 21, 23, 24]. In-band approaches are *invasive*: they consume limited resources that interfere with the primary tasks of sensing, storing, processing, and transmitting data.

Out-of-band management isolates management tasks on a separate always-on control processor and radio attached to each node. The approach divides WSN functions between a *data plane* that senses, stores, processes, and transmits data, and a *management plane* that ensures continuous visibility, accessibility, and interactivity. The division takes advantage of the natural distinction between

control traffic (short infrequent interaction) and data traffic (bulk data transfer). Out-of-band management is also a common technique for diagnosing and repairing node failures in other distributed systems, such as networks and data centers. While the energy costs of using a separate per-node control processor and radio preclude out-of-band management in some scenarios, the approach is well-suited for *high-power* WSNs that handle data from one or more high-bandwidth sensor streams and engage in computationally-intensive processing.

Examples of such high-power sensor applications include networks of weather sensing radars [14] and camera networks [19]. Our target application is monitoring river ecologies using a diverse array of connected sensors, including underwater camera, hydrophone, water quality, geologic imaging, and temperature sensors. Since high-power WSNs already require enough harvested energy to support a powerful node platform (*e.g.*, an iMote2, Gumstix, or embedded PC-class node), it is feasible to continuously operate a less powerful control processor and radio that uses a small fraction of the main node's power (*e.g.*, a TinyNode, TelosB, or MicaZ mote) and has a minimal impact on the data plane's operation.

To illustrate, Figure 1(a) provides an overview of our prototype's management plane, which uses a Tinynode with an XE1205 radio as the control processor and a Gumstix [1] with a PXA-based microcontroller and commodity 802.11b WiFi radio as the high-power main node. The XE1205's long range (1.43 miles at 4.8kbps [6]) makes it particularly attractive for out-of-band management. Importantly, the Tinynode control processor is also able to control the sensors directly without consuming additional energy by powering the main node. Figure 1(b) shows average energy production from an attached 4"x8" solar panel on a typical sunny summer day and cloudy winter day compared with the prototype's average energy consumption in different power states. The measurements demonstrate that even on a worst-case cloudy winter day the control processor is able to remain on continuously using a small amount of buffered energy from a battery, while the main node must remain mostly off due to the high energy cost of operating its processor, radio, and sensors.

Contributions. This paper presents *SRCP*, a Simple Remote Control Protocol for use on low-power control processors and radios that forms the core of a non-invasive out-of-band management plane for perpetual high-power WSNs. A key design principle of SRCP is to expose a narrow set of management primitives *without* defining management services; as a result, much of its power derives from connecting controllers to existing management functions provided by high-power hardware platforms and their software. SRCP's narrow set of primitives is able to unify a broad spectrum of management functions. In particular, we show that the protocol is able to monitor the health of the network at fine granularities (1 update every 250 milliseconds for a 5 hop network), support interactive debugging sessions using a low-bandwidth radio (less than 2 second latency per directive), and non-invasively transfer bulk software updates using a DTN routing protocol.

2 Related Work

In-band Management. In-band techniques improve visibility, accessibility, and interactivity within the confines of a node's energy constraints and duty cycle, and are orthogonal to SRCP, which assumes an always-on control processor and radio.

Sympathy [18] and PCP [21] improve visibility indirectly by correlating lack of sensor data from a node with failure and then diagnosing the root cause by traversing a decision tree of likely possibilities. NodeMD addresses visibility, by using a runtime detection system to detect faults, and interactivity, by catching many faults before they disable the system and enabling a debug mode [11]. Clairvoyant [24] and Marionette [23] also improve interactivity by enabling interactive debugging. In contrast, SRCP is able to connect operators to existing debuggers available for commodity OSes, such as GDB, to enable interactive debugging of data plane software.

Dissemination techniques for efficiently updating node software, including Trickle [13] and Deluge [10], improve accessibility. Since we target high-power WSNs using complete network stacks for SRCP, more general communication techniques, such as end-to-end TCP connections or DTN-style store-and-forward paradigms, are possible for both disseminating data plane software updates and transmitting sensor data. Finally, SNMS recognizes the importance of separating the management plane from the data plane by decoupling them to the extent possible using an in-band approach and implementing a broad set of management services [20]; SRCP completely decouples the two planes and provides narrow primitives for connecting to existing management services.

Out-of-band Management. Out-of-band management is a necessity in sensor testbeds that support multiple experiments over time, as in MoteLab [22] or Trio [7]. These testbeds utilize a back-channel or control processor to provide continuous visibility and access to nodes. However, the main purpose of the back-channel is to enforce the testbed's scheduling policies, ensuring that no experiment monopolizes testbed resources, and to deploy experiment-specific software. Perhaps most related to SRCP is the Deployment Support Network (DSN), which mitigates the need for a wired back-channel in testbeds by attaching a separate battery-powered control processor and radio, called a DSN-node, to each main testbed node [8].

In contrast to DSN, our target application is a perpetual deployment, which warrants a focus on a simple protocol suitable for low-power control processors and radios that adaptations to DSN-node software do not easily address. DSN-nodes implement common testbed services, such as event logging, interactive debugging, and software distribution, whereas SRCP is protocol-centric and simply enables remote access to software services and low-level hardware functions that already exist for high-power nodes. The choice of radio for DSN highlights the different focus: DSN-nodes use Bluetooth for their backbone wireless network while we choose the XE1205. The XE1205 is a low-power, long-range, and low-bandwidth radio that allows our management plane to operate over longer

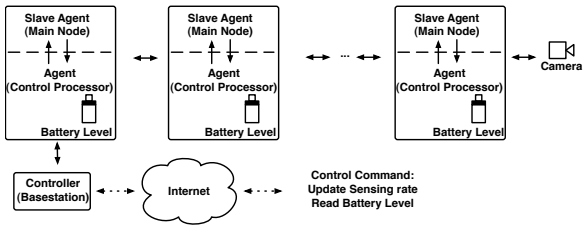


Fig. 2. An SRCP agent runs on each control processor and a slave agent runs on each main node.

distances than short-range Bluetooth radios, decreasing both the incidence of network partitions due to control processor failure—since it may be possible to "hop over" failed nodes—and the available bandwidth for the management plane. The XE1205 is also not invasive: it minimally impacts node energy consumption and eliminates traffic conflicts with a shorter range main node radio.

While SRCP does assume dual-radio/dual-processor nodes, previous work on these systems focuses primarily on dynamically assigning tasks to components to adjust the energy/performance ratio on specific hardware platforms [3, 5, 16]. SRCP's focus is more narrow but also more general: it defines a simple protocol that is a foundation for remote management of a broad range of hardware platforms with control processors. In particular, Leap is a hardware/software architecture using two battery-powered processors and radios that shares our focus on high-power WSNs [16]. Leap and SRCP share a common goal but a different focus: SRCP is a simple, extensible, and hardware-independent protocol that distills a small set of core remote management functions for always-on control processors, while Leap combines a hardware platform that supports fine-grained energy monitoring with algorithms for dynamically scheduling tasks to processors to minimize energy consumption.

3 SRCP: A Simple Remote Control Protocol

SRCP is inspired by SNMP, a standard for managing network-attached devices. SNMP's primary use is *monitoring* wired network devices, while SRCP's primary use is monitoring and *controlling* wireless network devices using an external control processor. Figure 2 shows an overview of SRCP's node architecture and software artifacts. Each control processor runs an instance of an SRCP *agent* that implements the protocol. A *slave agent* also runs on each main node and interacts with its agent to permit interaction with the main node's software services.

SRCP is a basis for both out-of-network and in-network control. With out-of-network control, an SRCP *controller* injects protocol messages into the management plane from a well-connected base station within range of at least one agent. For example, the controller may disseminate protocol messages that power down all main nodes during uninteresting periods (*e.g.*, at night in a camera network),

or power up main nodes on-demand during interesting periods of plentiful energy (*e.g.*, to apply synchronized software updates or initiate immediate sensor data collection and transmission).

With in-network control, one agent issues protocol messages to another agent to control its behavior. For example, an agent for a node with a full buffer of sensor data could issue a control message to activate an upstream node in order to transmit the data and free buffer space for additional sensing. The use of SRCP for in-network control requires inter-node cooperation to ensure that the decisions made by agents using their local knowledge result in acceptable global WSN-wide outcomes that efficiently use energy, network, processing, and storage resources. While the protocol does support in-network control, techniques for inter-node cooperation are outside the scope of this paper; instead, we focus on out-of-network control using a controller running at a base station.

3.1 Protocol

The protocol is based on short *control messages*, which may be fragmented into one or more transmitted *packets*. Each message is sent by an SRCP controller to one and only one agent, and directs the agent to take a specific *action* at a node that produces an *outcome*. The one-to-one communication paradigm is simple and enables management of each node as a distinct entity, rather than in-band approaches that expose WSNs as a single aggregate. Upon message receipt, the agent invokes the specified action and then transmits a *response message* to the controller that encapsulates an acknowledgement of control message receipt, an indication of the action's success or failure, and an action-specific payload that encodes a description of its outcome.

A key goal of SRCP is to separate remote management primitives from specific remote management services; the primitives are general enough to serve as a foundation for monitoring any hardware platform or controlling any software on the main node. As a result, SRCP is extensible since some actions require hardware or software support from the main node or attached sensors that may vary across platforms. Developers register hardware-related actions (Execution and State) prior to deployment by defining a unique index number and linking the action's logic to the implementation at compile time, while controllers are able to register other actions (Conditional and Connection) *in situ* post-deployment.

3.2 Primitives

The protocol distills actions into four fundamental classes of remote management primitives: Executions, States, Conditionals, and Connections. Each class consists of one or more distinct actions, where the controller and agent associate each action with a unique integer index. Control messages include this index, which also identifies the message class, as part of the message payload to direct the agent to act on a specific Execution, State, Conditional, or Connection.

Executions. An execution is an action that affects the operational state of

the main node or any attached sensors. In particular, executions make visible hardware/software control of the main node and attached sensors that would otherwise not be available when the main node is inactive and powered down. For instance, our reference implementation includes an execution action that directs the slave agent to invoke an arbitrary process on the main node and return its standard output and standard error in a response message. If the main node OS supports fine-grained resource control (*e.g.*, Resource Containers [4] or PixieOS tickets [15]), then execution actions may also dynamically control node energy, CPU, memory, or bandwidth usage. Other examples include:

- Power-on Main Node; Power-off Main Node; Sleep Main Node (ACPI S3); Hibernate Main Node (ACPI S4); Power-on Main Radio; Low-power Main Radio; Power-off Main Radio; Main Node Process Execution; Main Node File Transfer; Take Picture; Transfer Picture to Flash; Main Node Alive Ping; Reboot Main Node Standard Kernel; Reboot Main Node Clean Kernel;

The payload of the control message includes its index along with execution-specific data, while the payload of the response message includes details of the execution's outcome.

States. Actions may read, and in some cases write, state variables stored by the control processor using its limited on-board memory. SRCP divides state variables into two categories: environmental states and management states. The value of environmental state, such as the current battery level, is dictated by the environment and is read-only, while the value of management state, such as a routing table entry, is writeable remotely using a control message. Examples of environmental and management states include:

- **Environmental States.** Battery Energy Level; Solar Power Production; Platform Power Consumption; Main Node Reboot Counter; Control Processor Reboot Counter; Current Time;
- **Management States.** Voltage Level; Flash Memory (via JTAG); Main Node Processor Registers (via JTAG); Routing Table Entry; Conditional Period; Environmental State Update Period;

The agent automatically monitors and updates the value of environmental states at a predefined time granularity. The payload of a state-based control message includes its index, a flag indicating a read or write operation, and a value if applicable. The response message payload includes the value of the state and any acknowledgements.

Conditionals. In some cases, it is necessary for an agent to react immediately to local conditions or at a prespecified time. A Conditional action invokes an execution or state-based action based on a condition. The condition is a boolean expression composed of environmental or management states and boolean operators. The protocol supports two conditional types: one-time and continuous. A one-time Conditional action executes a single time when a condition is true,

while a continuous Conditional action executes every time a condition is true every configurable time period. The controller is able to use control messages to dynamically add, remove, or modify an action's condition. The payload of a Conditional control message includes an index, a flag indicating whether to set a new condition or delete an existing one, an Execution or State action to execute, and the condition. The response message simply acknowledges the action's success or failure. Note that once a conditional is set, the controller will receive asynchronous response messages associated with its Execution or State actions.

Connections. Long-lived interactive sessions between a controller and a main node require reliable end-to-end communication not possible using short control messages. To accommodate interactive sessions, the agent forwards packets marked as connection actions directly to its slave agent. These packets are opaque to the agent and are only interpreted by the slave agent; the intent is to support network layer tunneling and end-to-end connections between the controllers and slave agents, which may both implement complete network stacks.The payload of a connection packet includes its index along with opaque data interpreted by the slave agent (*e.g.*, TCP packets). The payload of the response message includes its index along with opaque data interpreted by the controller (*e.g.*, TCP acknowledgements).

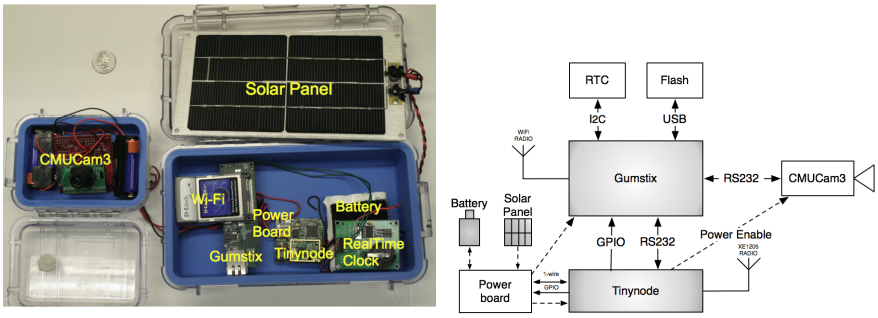
4 Implementation

We have written a reference implementation of SRCP [2]. While the protocol is platform-independent, the reference implementation is intended for a mote-class control processor with a Linux-capable main node and supports an agent for TinyOS and a slave agent for Linux. The implementation utilizes hardware features (*e.g.*, JTAG, numerous power states) that require a compatible hardware platform, although its core functions are portable to other platforms utilizing a TinyOS-based control processor and Linux-based main node.

4.1 Hardware Prototype

Since many of the protocol's functions interact with specific hardware features of the main node and sensors, we built a hardware prototype (Figure 3(a)) that fully utilizes it. The prototype is a general-purpose node platform for high-power WSNs; it uses a Tinynode control processor with a low-power MSP430 microcontroller, 512Kb of on-board flash, and an XE1205 radio. The XE1205 radio is attractive for out-of-band management since it does not interfere with the data plane's 802.11b radio, and is able to trade bandwidth for range. The measured range of the radio has been shown to be 2.3 kilometeres (1.43 miles) at a bandwidth of 4.8kbps, exceeding the range of the Mica2 or Telos by at least a factor of 4 [6]. Additionally, we also evaluated the implementation on TelosB motes using the more capable CC2420 radio.

[2] `http://lass.cs.umass.edu/`



(a) Hardware Prototype (b) Block Diagram

Fig. 3. A photograph (a) and a block diagram (b) of our hardware prototype.

The prototype uses a Gumstix with a PXA-based microcontroller and a commodity 802.11b WiFi radio for the main node. The Gumstix runs a instance of Linux that supports standard Linux utilities. We attach the CMUCam3 imaging sensor as a representative example of a high-bandwidth sensor [19]. The prototype also includes external flash for additional Gumstix storage and a real-time clock that the Gumstix and Tinynode use to periodically synchronize their notion of time. Figure 3(b) shows a block diagram of the prototype. Communication between the Tinynode's agent and the Gumstix's slave agent occurs over a serial RS-232 connection. The main powerboard regulates charging from energy produced by a SPE-350-6 SolarWorld solar panel, stores it in 3.7V Ultralife rechargeable battery with a capacity of 6.1 Amp-hours, and distributes it to the Tinynode, Gumstix, and CMUCam3 sensor. The materials for each prototype node cost approximately $650.

To increase management flexibility, the Tinynode is capable of independently controlling functions on the Gumstix, its WiFi radio, and the CMUCam3 sensor. A modular hardware platform, such as our prototype or LEAP [16], is a useful paradigm for high-power WSNs, since they allow a controller to independently power and operate each component. Our experiences demonstrate that SRCP is flexible enough to support a range of functions on different devices in such a platform.

4.2 Software Prototype

The SRCP reference implementation, described below, includes an agent written in NesC for TinyOS, a slave agent written in C for Linux, and a simple controller for a base station written in C.

– **Agent**. The agent implements the protocol from Section 3 and supports the example Execution and State actions from that section. We discuss details of network communication in our prototype (*e.g.*, routing, packet format) in Section 4.3.

– **Slave Agent**. The slave agent runs on Linux and integrates with the TinyOS serial forwarder to send and receive control messages from the agent. In addition to interpreting Connection messages, the slave agent includes support for specific Execution actions that integrate with software supported by the node. For instance, our implementation includes support for executing processes and receiving standard error and out using control messages, and direct file transmission over the management plane. Both actions are suitable for short-lived interactions (*e.g.*, quick process execution or small files), or as a "last resort" for connecting to the main nodes when communication via WiFi is impossible. As discussed in Section 5, our slave agent also incorporates a DTN routing reference implementation for disseminating bulk software updates and gathering sensor data.
– **Controller**. The controller includes a management shell and GUI dashboard to manually inject control messages and view their responses. Designing policies that dynamically adjust the WSN's behavior based on both environmental conditions and data requirements is the subject of future work. The base station includes both an 802.11b WiFi radio for data plane communication and a root Tinynode with an XE1205 radio running an agent connected over an RS-232 serial link. The controller integrates with the TinyOS serial forwarder to forward control messages over the serial link to a "first-hop" SRCP agent that routes them to their destination using the XE1205 radio.

IP Tunneling. To support TCP/IP flows between a controller and slave agent using Connection messages, both include a Control Message/IP proxy for establishing IP tunnels. The proxy (i) captures egress IP packets, fragments them into control or response messages, and forwards them to its agent for transmission over the management plane and (ii) reassembles ingress control messages into IP packets and injects them into the network stack.

The proxy currently utilizes the Linux Netfilter library for capturing egress packets from the network stack and Linux raw sockets for injecting ingress packets back into the network stack, although we are exploring the benefits of TUN/TAP-based implementation. We assign the base station and Gumstix nodes an IP address in the 172.16.0.0/16 subnet; the proxy then uses standard `iptables` rules to capture all packets destined for the subnet for tunneling. While SRCP could use the an implementation of 6LoWPAN to forward IPv6 packets without tunneling them, the XE1205 radio does not support 6LoWPAN's standard 127 byte packet size [17]. 6LoWPAN is not necessary for our prototype since the control processor does not serve as a connection end-point, and the Gumstix main node supports a complete network stack.

Since long-range radios cannot sustain high bit rates, the proxy includes an implementation of Van Jacobson header compression (IPcomp) from RFC 1144 to reduce the length of TCP/IP from 40 bytes to 1 or 2 bytes on average. IPcomp is useful for reducing overhead in interactive sessions composed of a series of small packet transmissions (*e.g.*, ASCII characters), where TCP/IP headers can consume up to 50% of a TCP packet's size. Section 5 quantifies the effect of IPcomp on interactivity. Additionally, we used suggestions from RFC 3150 to

Primitive	*Command*	*Power (μjoules)*	*Max*
Execution	Wakeup Gumstix	0.551	$1.47x10^{11}$
Set Conditional	Wakeup Gumstix in 5 minutes	0.580	$1.40x10^{11}$
Read State	Sensing rate	13.92	$5.84x10^{9}$
Write State	Sensing rate	13.97	$5.82x10^{9}$
Connection	Transmit 28 byte packet	0.560	$1.45x10^{11}$

Table 1. *Max* shows the maximum number of times the Tinynode command could be executed based on our prototype's battery with capacity 81, 360 joules.

set TCP parameters for low speed and unreliable lengths (*e.g.*, lowering TCP Maximum Segment Size (MSS) from its default of 1500 bytes).

4.3 Management Plane Communication

A radio for out-of-band management values transmission range and energy efficiency over bandwidth, since disconnected nodes impede visibility, accessibility, and interactivity. The XE1205 radio we choose for our prototype imposes limitations on the maximum possible packet size: the radio does not reliably support packets larger than 28 bytes (the default AM packet size in TinyOS) due to a 16 byte send/receive FIFO buffer that requires 50 μsec to empty [3]. As a result, our implementation imposes limits on header sizes and does not provide a reliable transport protocol, as discussed below.

Packet Format. Each control message packet includes a minimal header with fields for a message identifier, a sender's identifier, a destination's identifier, a fragment number, a message length, and a time-to-live value. Each identifier uniquely identifies the message, sender, and destination; the destination uses the fragment number and message length to reassemble the message; the time-to-live value defaults to the network's diameter and ensures that the network eventually drops packets that cannot reach their destination. We use 2 bytes for the message length field and a single byte for the remaining fields. Thus, our implementation uses 7 bytes out of each 28 byte packet for headers (25%) and 21 bytes for the payload (75%). The sender identifier, destination identifier, and time-to-live fields must increase to scale the protocol to networks larger than 255 nodes or with a diameter greater than 255.

Reliable Communication. While agents acknowledge messages using responses, as described in Section 3.1, the controller and agent do not acknowledge packets end-to-end, since acknowledgements at each level of the network stack consume bandwidth. The implementation uses only simple link-layer acknowledgments provided by TinyOS's AM abstraction to ensure reliable per-hop packet transmission. An agent retransmits each packet if it does not receive a link-layer acknowledgement within timeout t, and after k retransmissions it drops the packet. Without end-to-end packet acknowledgements, the loss of a single

[3] The time taken to empty the buffer 3 times per single packet reception or transmission results in unacceptable loss rates.

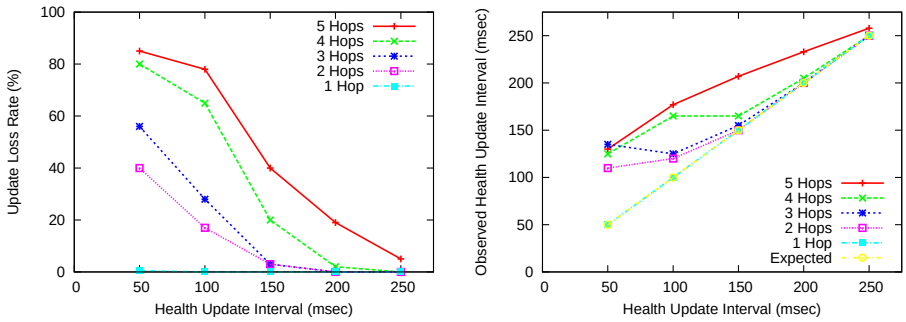


Fig. 4. Measurements of the percentage of update losses (a) using Conditional actions to monitor node health over a range of intervals for different network sizes. The observed health update interval (b) is the interval between health updates seen at the controller.

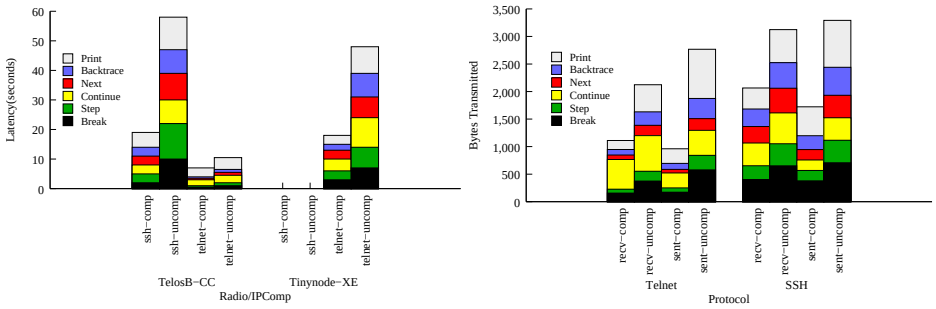
packet prevents the delivery of an entire control or response message. As a result, the implementation limits many control and response messages to a single packet, although the size of some messages, such as an encoded outcome in a response message, may be an arbitrary length.

Routing. Finally, agents must be able to route packets from the controller to the packet's destination. Our implementation assigns each agent a simple static identifier, and agents forward any received packet with a destination identifier that does not match its own to the next hop. Each agent maintains a routing table as special management states. To determine the next hop, the agent looks up the destination identifier in its set of management states, and interprets the value as the next hop identifier. The controller is able to alter routes dynamically using State actions, although we have not explored dynamic approaches to routing.

5 Evaluation

We evaluate SRCP with a case study that exemplifies the actions we envision a WSN controller using to manage a network, and quantifies SRCP's performance along three axis: visibility, interactivity, and accessibility.

Visibility. Wachs et al. define a visibility metric as the energy required to diagnose a failure [21]. Table 1 reports microbenchmarks of the CPU time and energy cost to a Tinynode in our prototype to execute an action from each class, and demonstrates that individual actions are non-invasive and impose little energy cost on the data plane. The primary energy cost derives from keeping the control processor active (see Figure 1(b)) and not from executing individual actions.



(a) Latency for GDB (b) Bytes Sent/Received for GDB

Fig. 5. Telnet sessions using IPcomp and 100 byte packets on the CC2420 radio perform best for interactive GDB debugging sessions (a). IPcomp has a factor of 2 impact on total bytes transmitted (b).

Since enough energy exists to continuously operate SRCP's control processor and radio, an energy-centric visibility metric is not appropriate. Instead, SRCP's primary constraint is management plane network bandwidth; as a result, we define visibility as the rate (messages/second) at which a controller is able to observe changes in the state of the network. The frequency of node health updates is a direct measure of visibility since they expedite the discovery of anomalies or failures.

In Figure 4(a), we observe the loss rate for node health updates as a function of their send rate. For the experiment, we use a configuration of 5 nodes in a simple chain topology, where each node is 15 meters from its neighbors. Each node uses a Conditional action that reports battery level and pings the main node for liveness; note that the health updates could include any of the management or environmental states listed in Section 4. The x-axis shows the health update interval for each node and the y-axis shows the percentage of updates lost in the management plane and not delivered to the controller. The result demonstrates that the prototype is able to sustain an update interval of 250 milliseconds in a 5 hop network without experiencing significant losses due to network congestion, allowing a controller to detect any node anomalies (*e.g.*, low battery level, failed main node) at a 250 millisecond granularity. In Figure 4(b), we plot, for the same experiment, the observed average health update interval seen at the controller, demonstrating that the observed rate is close to the expected interval even when experiencing congestion-related losses.

Our results indicate that the management plane should be able to monitor a network of N nodes at an interval of $250N/5 == 50N$ milliseconds; for a network of 100 nodes this translates to an update interval of 5 seconds. In practice, we expect the controller to observe the entire WSN at a coarse granularity and focus in on specific regions with a finer granularity once an anomaly is detected.

Interactivity. Operators must diagnose and repair problems in the data plane

that impair operation. Rather than indirectly diagnosing a problem, as in Sympathy [18], SRCP uses Connection actions to enable interactive debugging sessions on the main node. As with visibility, the primary constraint is network bandwidth. Figure 5(a) measures the latency for a representative interactive GDB session using a set of common debugging commands over both Telnet and SSH with and without IPcomp in a 5 hop network [4]. Since the XE1205 radio prevents packet sizes greater than 28 bytes, we also show results using TelosB motes with a CC2420 radio to study the impact of larger packet sizes on latency.

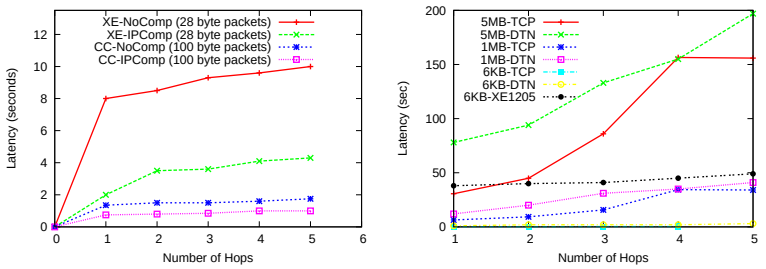
The measurements show that interactive sessions using Telnet and IPcomp are possible for both radios. However, the XE1205's 28 byte packet size limitation prevents tolerable interactive sessions for SSH with or without IPcomp. IPcomp has a significant impact on both Telnet and SSH, improving latency by at least a factor of 2 for all commands. Figure 5(b) shows total bytes sent and received for both sessions. In the best case—Telnet with IPcomp—the interactive latency is less than 3 seconds for each command for the XE1205 and less than a second for the TelosB. Figure 6(a) shows that the session latency increases modestly with the number of network hops; extrapolating the trend indicates that a 30 hop network path should experience sub-10sec latency for the total session with the XE1205 using Telnet/IPcomp.

Accessibility. Once an operator diagnoses a problem using an interactive debugging session it may be necessary to update the node's software. SRCP enables accessibility at each level of a node's hardware/software stack. At the lowest level, our implementation interacts with the Gumstix JTAG controller to provide remote access and control of its hardware, as proposed in [9].

The SRCP agent uses an execution action to expose remote access to JTAG through a set of 4 GPIO pins; in a conventional setting, these GPIO pins would connect to a PC through a USB or parallel-port JTAG connector. JTAG integration enables two capabilities: (i) direct control to read and write processor registers, including the instruction register, and clock the CPU and (ii) direct reading and writing of Flash memory. The first capability is useful for running hardware diagnostics on nodes without an operational OS, while the second capability is useful as a "last resort" for reading the state of flash off a failed node or writing flash directly to reconfigure a failed node from scratch (*e.g.*, install a new bootloader/minimal kernel). Our microbenchmarks show that it takes 205.3 seconds to write the Gumstix's 163kB uBoot bootloader to Flash one word at a time in a single hop network.

At the next level of the stack, our implementation integrates with the Gumstix's uBoot bootloader to implement a SafeBoot mechanism as an Execution action. SafeBoot allows the controller to select either a "safe" kernel or a standard kernel when rebooting a node. The safe kernel is preloaded on Flash and is read-only, while the controller may update and modify the standard kernel to

[4] The communication cost of the remote GDB protocol, which transfers individual assembly instructions, consumes enough network bandwidth to prevent tolerable interactive sessions.



(a) (b)

Fig. 6. Latency for interactive sessions increases modestly with the number of network hops (a). We use DTN (b) to opportunistically and non-invasively transfer bulk software updates in the data plane.

upgrade drivers or propagate patches. To initiate SafeBoot, the SRCP agent sets a GPIO pin on the Gumstix prior to applying power; the mechanism requires a minor modification to uBoot to check the pin state prior to boot to determine the appropriate kernel. A controller may also use the SafeBoot mechanism or the reboot mechanism in conjunction with Conditional actions to implement watchdog or grenade timers that periodically bring nodes to a clean state.

At application-level, our slave agent incorporates the reference implementation of DTN2 for non-invasive bulk software updates [2]. Software updates represent the one area where the management plane, due to bandwidth limitations, leverages the data plane for tasks that are not data-centric. Rather than requiring the controller to coordinate activation of every upstream node to update a downstream node's software using direct TCP connections, which would impact the operation of the data plane, we use DTN to opportunistically route data as main nodes become active. Figure 6(b) compares the latency to transfer different size files over DTN and TCP in the data plane and using an SRCP Execution action in the management plane. The benchmark demonstrates that the management plane is not suitable for software updates or other bulk data transfers (6kb takes 38sec over a single hop), and that, while not performing as well as TCP, DTN is a useful tool for non-invasive bulk data transfer over the management plane (5MB takes 200sec over 5 hops).

6 Conclusion

The energy demands of emerging high-power WSNs permit non-invasive out-of-band management through an always-on control processor powered by harvested energy. SRCP enables the paradigm using agents to monitor or change a node's operational, environmental, and management state and connect to its software services. Our evaluation shows that SRCP's primitives unify a broad range of management functions, and its performance is acceptable and non-invasive. .

References

1. Gumstix. http://www.gumstix.com, Accessed February 2008.
2. Delay Tolerant Networking Research Group. http://www.dtnrg.org/wiki/Code, Accessed February 2008.
3. Paramvir Bahl, Atul Adya, Jitendra Padhye and Alec Walman. Reconsidering Wireless Systems with Multiple Radios. In *SIGCOMM Computer Communications Review*, 34(5):39-46, October 2004.
4. Gaurav Banga, Peter Druschel and Jeffrey C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In Proceedings of the *Symposium on Operating System Design and Implementation*, pages 45-58, New Orleans, Louisiana, February 1999.
5. Richard Draves, Jitendra Padhye and Brian Zill. Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks. In Proceedings of the *International Conference on Mobile Computing and Networking*, pages 114-128, September 2004.
6. Henri Dubois-Ferriére, Laurent Fabre, Roger Meier and Pierre Metrailler. TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications. In Proceedings of the *International Conference on Information Processing in Sensor Networks*, pages 358-365, Nashville, Tennessee, April 2006.
7. Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Cory Sharp, Jay Taneja, Gilman Tolle, Kamin Whitehouse and David Culler. Trio: Enabling Sustainable and Scalable Outdoor Wireless Sensor Network Deployments. In Proceedings of the *International Conference on Information Processing in Sensor Networks*, pages 407-415, Nashville, Tennessee, April 2006.
8. Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin and Philipp Blum. Deployment Support Network - a Toolkit for the Development of WSNs. In Proceedings of the *European Conference on Wireless Sensor Networks*, January 2007.
9. Hans Eberle, Arvinderpal Wander and Nils Gura. Testing Systems Wirelessly. In Proceedings of the *IEEE VLSI Test Symposium*, pages 335, April 2004.
10. Jonathan W. Hui and David Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming At Scale. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, pages 81-94, Baltimore, Maryland, November 2004.
11. Veljko Krunic, Éric Trumpler and Richard Han. NodeMD: Diagnosing Node-Level Faults in Remote Wireless Sensor Systems. In Proceedings of the *International Conference on Mobile Systems, Applications, and Services*, pages 43-56, San Juan, Puerto Rico, June 2007.
12. Philip Levis, Eric Brewer, David Culler, David Gay, Samuel Madden, Neil Patel, Joe Polastre, Scott Shenker, Robert Szewczyk and Alec Woo. The Emergence of a Networking Primitive in Wireless Sensor Networks. In *Communications of the ACM*, 51(7), July 2008.
13. Philip Levis, Neil Patel, David E. Culler and Scott Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In Proceedings of the *Symposium on Networked System Design and Implementation*, pages 15-28, San Francisco, California, March 2004.
14. Ming Li, Tingxin Yan, Deepak Ganesan, Eric Lyons, Prashant Shenoy, Arun Venkataramani and Michael Zink. Multi-User Data Sharing in Radar Sensor Networks. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, pages 247-260, Raleigh, November 2007.
15. Konrad Lorincz, Bor-rong Chen, Jason Waterman, Geoff Werner-Allen and Matt Welsh. Resource Aware Programming in the Pixie OS. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, 2008.
16. Dustin McIntire, Kei Ho, Bernie Yip, Amarjeet Singh, Winston Wu and William J. Kaiser. Low Power Energy Aware Processing (LEAP) Embedded Networked Sensor System. In Proceedings of the *International Conference on Information Processing in Sensor Networks*, pages 449-457, Nashville, Tennessee, April 2006.
17. Geoff Mulligan. The 6LoWPAN Architecture. In Proceedings of the *Workshop on Embedded Networked Sensor*, pages 78-82, June 2007.
18. Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler and Deborah Estrin. Sympathy for the Sensor Network Debugger. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, pages 255-267, San Diego, California, November 2005.
19. Anthony Rowe, Charles Rosenberg and Illah Nourbakhsh. A Low Cost Embedded Color Vision System. In Proceedings of the *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 208-213, October 2002.
20. Gilman Tolle and David Culler. Design of an Application-Cooperative Management System for Wireless Sensor Networks. In Proceedings of the *European Workshop on Wireless Sensor Networks*, pages 121-132, January 2005.
21. Megan Wachs, Jung Il Choi, Jung Woo Lee, Kannan Srinivasan, Zhe Chen, Mayank Jain and Philip Levis. Visibility: A New Metric for Protocol Design. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, pages 73-86, Raleigh, November 2007.
22. Geoffrey Werner-Allen, Patrick Swieskowski and Matt Welsh. MoteLab: A Wireless Sensor Network Testbed. In Proceedings of the *International Conference on Information Processing In Sensor Networks*, pages 483-488, Los Angeles, California, April 2005.
23. Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta and David Culler. Marionette: Using RPC for Interactive Development and Debugging of Wireless Embedded Networks. In Proceedings of the *International Conference on Information Processing in Sensor Networks*, pages 416-423, Nashville, Tennessee, April 2006.
24. Jing Yang, Mary Lou Soffa, Leo Selavo and Kamin Whitehouse. Clairvoyant: A Comprehensive Source-Level Debugger for Wireless Sensor Networks. In Proceedings of the *Conference on Embedded Networked Sensor Systems*, pages 189-203, Raleigh, November 2007.