A Foundation for Interoperable Sensor Networks with Internet Bridging *

Alan Marchiori, Qi Han
Department of Mathematical and Computer Sciences
Colorado School of Mines
Golden, CO 80401
{amarchio,qhan}@mines.edu

Abstract

Current sensor networks tend to have a solitary focus where the entire sensor network is dedicated to solving a single problem. We believe this approach suppresses the wide-spread acceptance of sensor networks because each network needs to be individually designed and configured. This is in contrast to the Internet, where numerous systems of vastly different design successfully share the network medium without the need for custom configuration. In this paper we present a simple framework that provides basic network services for resource-constrained nodes. This framework has interfaces similar to Internet style protocols. This ensures that (1) different sensor nodes can interoperate without limiting the applications of sensor networks; and (2) integrating sensor networks to the Internet can be implemented easily with standard networking protocols. We use a simple environmental monitoring application to demonstrate that the proposed solution performs nearly as well as an implementation using standard TinyOS components in terms of packet delivery ratio. Furthermore, our approach provides the additional features of interoperability and seamless integration to the Internet while requiring 20% less ROM and 40% less RAM.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Wireless Sensor Network, Internet Bridging, Network Architectures

*This material is based upon work supported by the National Science Foundation under Grant No. 0720875.

EmNets'08, June 2–3, 2008, Charlottesville, Virginia, USA

1 Introduction

Wireless sensor networks have been rapidly evolving. Many current sensor networks are intelligent, autonomous, fault-tolerant, and full distributed. But, even in these highly advanced sensor networks, basic challenges do exist. Most sensor networks assume a homogeneous set of nodes serving a solitary purpose. This is contrary to the most common network, the Internet, where you can see various hardware and software platforms communicating and sharing data. What enables a computer to connect to an unknown network, send, and receive data without any manual configuration or protocol modifications? The answer is a mature networking stack that gives enough basic functionality to the application developer, without imposing many restrictions, namely TCP, UDP, IP, and DHCP. We could simply apply the same standard protocols to sensor networks. However, this would exclude many existing sensor nodes without sufficient memory or processing power to implement them. In addition, we would still need to develop methods to deploy, manage, and control this huge number of IP-based sensor nodes.

The approach taken in this paper is to define a networking stack that provides a basic set of services tailored to highly resource constrained wireless sensor nodes. These services must be simple enough to run efficiently on low-powered hardware and should not impose specific hardware or structural limits on the application. At the highest level, our networking framework should have some parallels with Internet networking. This will enable the two networks to be bridged without much difficulty. By exposing the sensor network to the Internet, we can easily develop processing and control applications that run on desktop computers with far more resources and more powerful development tools than the embedded sensors themselves. This will allow searching, control, data exchange, and management of sensor nodes from anywhere in the world.

2 Related Work

Existing sensor network architectures tend to impose too many or too few restrictions on the application designer. With too many restrictions the applications are

generally driven by the architecture; relying entirely on the architecture for every operation. With such restrictions, applications cannot reach beyond the provided architecture to provide truly innovative solutions. For example, Tenet [4] provides a tiered architecture for wireless sensor networks. Tenet, however, relies on a common task library to build applications. How can a single task library implement every possible application? In contrast, our architecture does not provide any application logic at all, only networking logic and a set of metadata descriptors that define each node's capabilities in a standard way. Application developers are free to implement application logic using any available programming language and style.

On the other hand, architectures with few restrictions may work well in isolation, however, when several applications are run on the same network, at the same time, the best case is usually that they totally ignore each other. The worse case is they interfere with each other and none of the applications work properly. This can be seen with TinyOS [2]. TinyOS is an operating system for wireless sensor nodes that includes many networking components to use as building blocks that may be used in any arbitrary configuration. While this provides the application developer with much flexibility, it also makes it very difficult if not impossible for different applications to interoperate. In contrast, nodes using our framework are able to communicate easily.

Another class of sensor network architectures is based on query processing such as TinyDB [6], SINA [8], Hourglass [9] and [1, 7]. These architectures view the sensor node as a data source and focus their efforts on efficient ways to extract the sensed data. We do not want to limit the domain of sensor networks to only sensing tasks. Our framework is extensible in that an existing query processing architecture such as TinyDB can be used in conjunction with our basic networking framework. This allows nodes both with and without a query processing service to share the same network.

3 System Architecture

An Internet connected sensor network can be considered a three-tiered network, depicted in Figure 1. The lowest tier contains the sensor nodes. Not all sensor nodes are identical, but are resource constrained. An example of a sensor node is the Berkeley Mote family. These low-level (LL) nodes are peripherals that interact with the environment and do little processing or decision making on their own. The next higher tier is called the intermediate-level (IL). This layer bridges the low-level nodes to a larger network, such as the Internet. The intermediate tier stores routing information for each node in the low-level network and routes messages from the high-level network to the low-level network. Examples of the intermediate tier are the Crossbow IMote2, Stargate, and the SunSPOT. Finally, the high-level (HL) tier interacts with the low-level through an intermediate-level node. This could be a result of a user accessing a web page, or a server controlling the nodes as part of an automated system. In order to support even the most

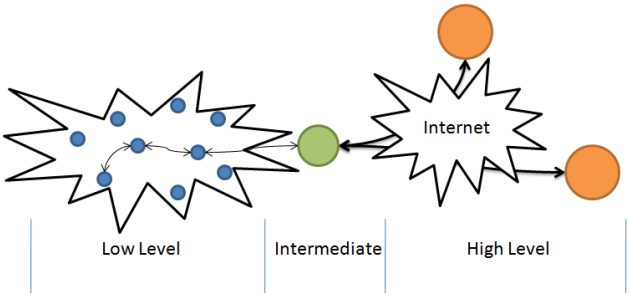

Figure 1. A tiered network architecture

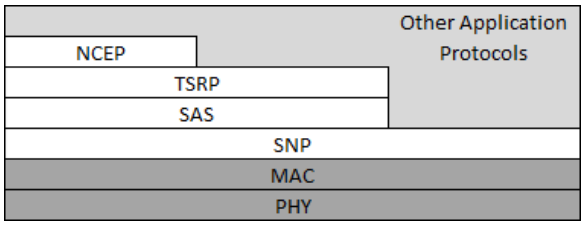

Figure 2. A complete protocol stack for resource constrained nodes

resource constrained low level nodes, our protocol stack is designed to be as simple as possible. In light of this, every low level node is not expected to perform complex operations such as in-network aggregation. Rather, they should just provide the most basic functions, such as reporting sensed data, responding to application requests, joining or leaving the network, and routing data from or to the base station. Applications that require more complex operations, however, can implement this functionality along side the basic protocol stack. To this end, we developed the following network stack for low level nodes (Figure 2). The Node Control and Event Protocol (NCEP) provides a simple interface for applications to interact with the low level nodes, allowing LL nodes to push information to HL nodes and HL nodes to pull information from LL nodes. The Tree Source Routing Protocol (TSRP) provides reliable multi-hop communication at the low level. To allow LL nodes to dynamically join and leave the network, the Simple Association Service (SAS) provides simple node membership management as well as supplying routing information to TSRP. To hide the complexity of platform specific MAC and PHY layers, a thin layer, the Sensor Network Protocol (SNP), is in place to multiplex higher protocols onto the radio MAC. In this framework, application logic can be implemented at any layer of the stack. Extensibility has also been incorporated into each layer of the protocol stack. By using port numbers, many protocols can be multiplexed onto the same network without interference.

Bytes				
1	1	1	1	n
Type=Event[1]	Length	Handle		Event Data
Type=Request[2]	Length	OpCode	Parameters & Request Data	
Type=Resposne[3]	Length	OpCode	Parameters & Response Data	

Figure 3. NCEP Packets

3.1 Node Control and Event Protocol

The Node Control and Event Protocol (NCEP) provides a bidirectional interface to low level nodes, similar to how an application can access hardware peripherals through a device driver. Three types of messages are defined in NCEP: event, request, and response (Figure 3). Request and response messages contain an op code; the valid op codes are listed in Table 1. 'Read' allows the LL node's metadata to be read. To interact with an LL node the 'call' op code is used. Calls are sent to specific peripherals using the handle parameter. The function identifier specifies which function of the peripheral is to be called.

Op Code	Parameters	Data
read	address, length	none
call	handle, function identifier	data

Table 1. NCEP Op Codes, parameters, and data

Metadata is used to identify the LL node and its peripherals. Table 2 shows the metadata contents. The peripheral count specifies the number of peripherals. The first peripheral descriptor (Table 3) follows immediately after the metadata, the remaining descriptors follow contiguously from this location.

Address	Length (bytes)	Description
0x00	8	Hardware ID
0x08	8	Software ID
0x10	12	Device Name
0x1c	2	Peripheral Count
0x1e	2	Device Specific

Table 2. Metadata contents

Address	Length (bytes)	Description
0x20	2	Manufacturer ID
0x22	2	Hardware ID
0x24	2	Driver Revision
0x26	2	Handle
0x28	8	Name

Table 3. Metadata peripheral descriptor

3.2 Simple Association Service

The Simple Association Service (SAS) allows LL nodes to join and leave the network. There are three types of SAS packets (Figure 4). At startup a low-level node begins in the unassociated state. While in the unassociated state, SAS request packets are broadcast periodically.

If a node receives an SAS request packet and is itself unassociated, it ignores the packet. A node that is associated, or is an IL node, will send a response message. Because multiple nodes may respond, a random delay is applied before sending a response. The response message contains the hop count to the IL node, link quality of the received request message, and the address of the IL node. If the IL node itself is responding, the hop count is set to zero.

Upon receiving an SAS response, the LL node extracts the hop count, link quality, root address, and sender's address from the message. If the node is not already associated (or this is the first response heard) the node saves the hop count and node addresses and enters the associated state. If the node is already associated, the message is still processed, as it might describe a better link. A response with a lower hop count, or equal hop count with higher link quality will overwrite the node's current association information with the new response's.

The notification message is broadcast in two cases: when a node's association changes or when it is about to leave the network. The sub-type field is used to identify these two cases. If a node receives a leaving message from the node it is currently associated with, it must send its own notification message and re-associate to the network. An association message is handled much like an SAS response, without the link quality field (although the local link quality may be used instead). If the hop count is lower than the node's current hop count, the node sending the notification is used as the next hop to the root node.

Because the SAS protocol operates locally, the higher level nodes may not know when an LL node joins the network. Therefore, it is the node's responsibility to send an NCEP event message after associating. The route this message takes is stored by the IL node and used to route messages back to that node.

3.3 Tree Source Routing Protocol

The tree source route protocol (TSRP) is designed to be a simple multi-hop routing protocol used to reliably deliver messages to and from the LL node. TSRP is much like a simplified version of TCP. It implements both link level and end-to-end acknowledgments, resends, and duplicate packet detection. Unlike TCP, it is connectionless to reduce the number of messages sent and does not provide fragmentation, congestion control, or flow control to reduce complexity.

Multi-hop routing to the root node is possible by using information from the SAS association process. Each node keeps track of the next hop toward the root node. Nodes closer to the root forward packets from distant

Bytes			
1	1	1	*AddrLen*
Request [0]			
Response [1]	Hop Count	Link Quality	Root Address
Notification [2]	Sub Type	Hop Count	Root Address

Figure 4. SAS Packets, where *AddrLen* is the length of a MAC address in bytes

Bytes					
1	1	1	1	$(AddrLen) * (HOPCOUNT + 1)$	n
Hop Count	Last Hop	Port	Seq. Num.	Route	Payload

Figure 5. TSRP Packet

nodes. This forms a routing tree to the root node in the network. Tree routing can be implemented easily and requires little overhead, which fits our design principle well: keep it simple.

To implement bidirectional communication using tree routing, each node would have to store the next hop to every other node in the network. To avoid this, when an LL node sends a packet to the IL node using tree routing, the IL node stores the route the packet took. When the IL has to send a message to that LL node it reverses this path and encodes the entire route in the message header. Each LL node looks at the hop count of the message and using the encoded route determines what the next hop should be. This is referred to as source routing, because the source node encodes the entire route. Intermediate nodes just have to examine the packet header to determine the next hop.

By using tree routing from the LL to IL and source routing from IL to LL, TSRP achieves a simple multihop routing system. The packet structure is shown in Figure 5. 'Hop Count' is the total umber of hops to reach the ultimate destination. 'Last Hop' starts at one and is incremented by one at each hop. At the final destination *Last Hop* = *Hop Count*. The field 'Port' is used to differentiate between different protocols implemented at this layer. TSRP also includes a sequence number to support retransmissions with duplicate packet detection. Finally, there is a list of MAC addresses representing the route. There are a total of *Hop Count + 1* addresses listed. The first address is the original source, then each hop, and finally the ultimate destination. For example, a 2-hop route from A to C through B would be encoded as {A,B,C}.

3.4 Sensor Network Protocol

The sensor network protocol (SNP) is a simple multiplexing protocol that sits above the radio MAC. This allows multiple protocols to send and receive packets at the lowest level of the network stack. Figure 6 shows the single header field, a one-byte port identifier. This protocol is the same as TinyOS's active message [5].

3.5 Internet Bridging

Bridging sensor networks to the Internet poses some difficult challenges. How do we search for and find sensors? How do we communicate with the sensor in a standard way? Our approach is to use an existing network protocol, Universal Plug and Play (UPnP) [3], that has many similarities to NCEP so it is easy to bridge the two protocols. UPnP has five parts: addressing, discovery, description, control, and eventing. Addressing is provided by the operating system while the remaining parts parallel the functions of NCEP closely.

Bytes	
1	n
Port	Payload

Figure 6. SNP Packet

Our proposed solution is to have the intermediate-level node create a virtual UPnP device representing each low-level node associated with it. The IL node would examine the LL node's metadata and then generate the corresponding UPnP device and service descriptions. These would be advertised by the IL node using the UPnP discovery protocol. If a node on the network wanted to access the LL node it would invoke a UPnP action which would be translated by the IL node to a NCEP call and sent to the LL node. The result would be collected and converted to a UPnP action response. Similarly, UPnP events will be generated by the IL node when a NCEP event message is received and any UPnP clients subscribed would also receive the event.

4 Performance Evaluation

To demonstrate the viability of our framework we have implemented it as a set of TinyOS [2] components. To perform an initial evaluation we have created a hypothetical temperature monitoring application. Each node simulates a temperature sensor that is allowed to increase or decrease by up to 5 degrees per second. If the temperature goes above a variable threshold, a message is sent to the root node containing the current temperature. No additional messages are sent until the temperature falls below the threshold and then again goes over the threshold.

This application was implemented using NCEP event messages to report high temperatures and NCEP commands to set the threshold. Another implementation using standard TinyOS components is used for compari-

son, where the TinyOS Collection component was used to report high temperature messages and the Dissemination service was used to set the temperature threshold. Table 4 shows the memory requirements of both implementations. Our proposed framework requires over 20% less ROM and 40% less RAM than the implementation using TinyOS standard components.

	Standard	Proposed
ROM	20.7 Kb	16.2 Kb
RAM	1.5 Kb	0.9 Kb

Table 4. Requirements using the MicaZ platform

To evaluate packet delivery performance, five different topologies were created each with 25 nodes randomly placed in a 25m by 25m area; one node simulated the IL node. The simulation was run until each node sent 25 temperature alert messages and repeated for each topology. The average results are shown in Table 5. The number of temperature alert messages sent is 600 for both cases. In terms of the number of unique alert messages received by the root node, these results demonstrate that NCEP is nearly as good at delivering packets as the collection protocol. Additional improvements could be made to NCEP by using link quality estimates provided by most radios. The TinyOS simulator, TOSSIM, does not implement link quality at the radio level. Using link quality in the SAS association process would likely improve the packet delivery rates.

	Standard	Proposed
Alerts Sent	600	600
Alerts Rcvd	599.6 (99.9%)	596.4 (99.4%)

Table 5. Packet delivery results

For the implementation using TinyOS standard components, we needed to post-process the results to remove duplicate alert messages. Before this post-processing, the average number of alerts received was 702.8. In the proposed solution there were no duplicate alert messages received due to the duplicate packet detection provided by TSRP.

The proposed solution also provides an acknowledgment back to the source node when the root node receives the alert message. The average number of acknowledged alert messages was 516.2. Lost acknowledgments are partially due to asymmetric links in the network. TSRP relies on symmetric links for bidirectional communication, so any asymmetric link will cause difficulty. This is a well known problem in sensor networks and we choose to accept this performance penalty to keep our implementation simple.

5 Conclusions

This paper presents a basic protocol suite for low-level sensing nodes focusing on simplicity, extensibility, interoperability, and Internet connectivity. A temperature monitoring application was simulated using our proposed approach and compared to a solution using standard TinyOS components. The result was significantly less memory usage while achieving similar packet delivery performance with the potential to bridge the sensor network to the Internet. The protocols presented in this paper builds a solid foundation for integrating wireless sensor networks with the Internet. Our next step is to implement the UPnP bridge node.

6 References

[1] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *Mobile Data Management*, pages 198–205. IEEE, 2007.

[2] David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo. A network-centric approach to embedded software for tiny devices. *Lecture Notes in Computer Science*, 2211:114–130, 2001.

[3] UPnP Forum. Upnp device architecture, `http://upnp.org/specs/arch/UPnP-DeviceArchitecture-v1.0.pdf`, access date: January 2008.

[4] Omprakash Gnawali, Ki-Young Jang, Jeongyeup Paek, Marcos Vieira, Ramesh Govindan, Ben Greenstein, August Joki, Deborah Estrin, and Eddie Kohler. The tenet architecture for tiered sensor networks. In *SenSys*. ACM, 2006.

[5] Jason Hill, Philip Bounadonna, and David Culler. Active message communication for tiny networked sensors, `http://www.tinyos.net/papers/ammote.pdf`, access date: January 2008.

[6] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[7] Marco Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli, and Jan Rabaey. A service-based universal application interface for ad-hoc wireless sensor networks (draft). November 2003.

[8] Chien-Chung Shen, Chavalit Srisathapornphat, and Chaiporn Jaikaeo. Sensor Information Networking Architecture and Applications. *IEEE Personel Communication Magazine*, 8(4):52–59, August 2001.

[9] Jeffrey Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, and Matt Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical report, 2004.