Reliability in Wireless Sensor Networks via a Modified TCP Implementation

Chris Blades
Department of Mathematics and Computer Science
Western Carolina University
chrisdblades@gmail.com

Joseph Randall Hunt
Department of Mathematics and Computer Science
Western Carolina University
randallhunt@gmail.com

Benjamin Rudolph
Department of Mathematics and Computer Science
Western Carolina University
bcrudolph1@catamount.wcu.edu

ABSTRACT

When wireless sensor networks first came out TCP was considered by most researchers to be contrary to the needs of wireless sensor networking. However, as the field has matured and common practices have been identified a need for reliable communication has become clear. This paper describes transmission control protocol implementations in wireless sensor networks and discusses the development of this particular implementation built on AMSend. The motivation behind this project is to create a reliable communications protocol for the telosb platform. This project attempts to solve the problems of reliable wireless communication using a variant of TCP (Transmission Control Protocol). The project, while unfinished, adds a reliable component to transmissions. Reliable communications are an important part of any WSN (Wireless Sensor Network) and should be of interest to both programmers and end-users of WSNs.

Categories and Subject Descriptors

C.2.1 [**Computer-Communications Networks**]: Network Architecture and Design—*Wireless, communication*; C.2.2 [**Computer-Communications Networks**]: Network Protocols; C.2.6 [**Computer-Communications Networks**]: Internetworking—*Standards*

General Terms

Wireless, Sensor, Networks, Design, TCP

Keywords

wireless, sensor networks, TCP

1. INTRODUCTION

Throughout the development of WSNs, reliable communications has been important component to the design of any system. The main problem this project addresses is that of reliable wireless communication at a software level. In a traditional wireless sensor network there is relatively little low-level information assurance. [4] Information assurance in this case has two meanings: both that the information is from an authenticated mote, and that the information is complete (undamaged in transit). This problem is important to solve because it affects all aspects of wireless sensor networking. At first glance the problem may seem trivial, but it is not, because of the platform's embedded nature. One must carefully consider how much power a system will take to maintain, how many extra messages need to be sent, and whether or not those messages will just generate more congestion in an already constrained airspace.

Implementation Goals

The system implemented by this group began with two goals: to implement an acknowledgement message and to create session management. ACKing is now implemented, but because of time constraints session management is not. The time involved also limited high-level empirical testing to determine efficacy of the implementation. Several of the problems that held up development are mentioned in section 2.2.

Paper Organization

Sections 2.1 and 2.2 present this modified TCP implementation and a test application, as well as an evaluation of this work. Section 2.3 discusses other implementations of reliable communication and compares them to this one. Finally, section 3 concludes in stating lessons learned while developing this project.

2. WIRELESS SENSOR TRANSMISSION CONTROL PROTOCOL

2.1 System Implementation

The traditional TinyOS Active Messages are based on the IEEE 802.15.4 16 bit short address and can only be allocated at program download time [1]. This leaves much to be desired and this implementation, though admittedly small, already improves on this. Traditionally TinyOS seems to operate on the Link layer as we can see in the message structure:

```
typedef nx_struct message_t {
    nx_uint8_t header[sizeof(
        message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(
        message_header_t)];
    nx_uint8_t metadata[sizeof(
        message_metadata_t)];
} message_t;
```

Listing 1: tos/types/message.h

This implementation of TCP was designed not to be fully standards-compliant, but rather small enough to operate efficiently. In this implementation of TCP for motes, the method was to provide a “wrapper” for reliable communications for any message struct (provided it is small enough) that the user is using. In cooperation with this wrapper, AMSend and Receive were both provided by the TCPSendC component. For the low level networking functionality, the original AMSend and Receive were used as Send and AMReceive. In doing this, it was possible to gain some of the reliability of TCP (through ACKing), while leaving it to the user to decide what to do in certain error conditions.

```
typedef nx_struct TCPMessage {
        nx_uint16_t ACK;
        nx_uint16_t fromAddress;
        nx_uint16_t payloadLength;
        nx_uint8_t payload[50];
} TCPMessage;
```

Listing 2: TCPMessage.h

The TCPMessage wrapper contains three 16-bit values: an ACK, the address the message was sent from, and the user message size. Finally, it contains a 50 byte array to store the user message. The ACK is a boolean TRUE or FALSE value (we did not implement sequencing, windowing, etc.). It is TRUE if the message being sent is an acknowledgement, FALSE if the message is simply an outstanding packet. The user message size facilitates extracting the user’s data intact without requiring excessive lengthy syntax on their part. The payload array is 50 bytes long, and is the most efficient way to store general data without knowing what it is, given that nesC does not really have inheritance or classes. It should be noted that the ACK could also be a much smaller data type, though the smallest known addressable space is a byte.

The Test Application

The test application uses a simple, 14 byte long SensorMessage struct as the user type. However, something much more complicated can be stored, given that the payload is simply 50 bytes of allocated memory. It uses the typical AMSend.getPayload(&message, sizeof(SensorMessage)), but it uses the implementation provided by TCPSendC. This loads their message into the TCPMessage’s payload array, and stores the user data size. Then, in the test application, the user message is stuffed with certain values for each field. It then sends this message off with the provided AMSend. The user must then decide what to do upon finishing sending (i.e. toggle an LED and unlock the message for editing).

Beneath the surface, TCPSendC also keeps track of how many times it has tried to retransmit the message. It sets a one shot timer upon transmission of a message. At a particular time after receiving no ACK (which it will know because the message will still be locked in the TCPSendC module), it will retransmit. After MAX_RETRY retransmits (set to 5 by default) it signals a sendDone(message, TCP_NOACK). It is up to the user to decide what to do at this point.

On the receive side, if it is the originally transmitting mote receiving an ACK, the message is unlocked and AMSend.sendDone(message, SUCCESS) is signaled. If it is the mote receiving the outstanding message, ACK is set to TRUE, the acknowledgement is sent back to the mote the message came from, and the provided receive event is signaled. In either case, double buffering is implemented.

For debugging purposes, LEDs were toggled at key events and data was watched using serial transmission.

2.2 Evaluation

Our work could have been better, but was sufficient, given the number of problems we ran into a long the way. Below are the problems and how they affected the development process.

Debugging

The first problem was the lack of any useful debugging tools. For one, there is no easy way to run a tool like gdb on the motes. The logic of a state machine is very different than that of the programming languages with which these students had more familiarity. These two aspects lead to a lot of difficulty in finding errors when they occur. There is no easy way to get data off the motes as they are executing code. Even the printf functionality doesn’t work as advertised in the documentation. This illustrates another problem in development of nesC programs.

Documentation

There was relatively little documentation other than the API and when there was useful documentation it was not consistent with documentation elsewhere. This lead us to empirical testing and a kind of trial and error approach.

New Language

This group was relatively unfamiliar with the language, which led to many time consuming errors (in overlooking race conditions and such) that an experienced nesC developer would not have had to deal with.

Further Questions

Other topics that would have been interesting to explore were the use of tasks and how this would have improved or degraded performance of the system. Given more time the team would have liked to answer several more questions about this particular project:

1. How does the system behave in low power mode?
2. At what distance does communication become completely unreliable?
3. How standards compliant should one be?
4. What parts of TCP should NOT be implemented?
5. What are the actual comparisons of this TCP implementation to regular transmission.

Question one is of particular concern because this is how most field applications of sensor networks operate. Obviously, with increased radio usage for ACKs the power consumption would be greater, but would it be great enough that operating the device in low power mode would be inadvisable?

Also, at what distance does communication become completely unreliable? Does this change with the use of TCP? The group estimates that TCP would dramatically extend the range at which one could communicate reliably.

Questions three and four are interesting, for a traditional implementation of TCP is not beneficial to WSNs. So, how does one decide what to include and what to exclude? A lot of the ramping up and ramping down for packet transmission is essentially useless for WSNs. IEEE specifies several standards for wireless communication–do WSNs fall into this realm? If so, which standard should apply? This is one area the team did not explore.

Finally, it would be good to get some empirical data supporting the use of TCP over other algorithms. Given the time constraints, we were unable to run any experiments though we did design them. One such experiment would be to have four motes transmitting a specified packet of data over a specified distance and keeping track of how many transmissions dropped and the total time it took for the data to be transmitted. One could adjust the distance and packet structure to measure the strengths and weakness of each protocol or algorithm.

2.3 Related Work

Of all the projects covered, the one that compares best to our work is something by David E. Culler at the University of Berkley [1]. In his talk he describes why UDP, a datagram transport protocol, fits so much better than TCP on the motes, as it's essentially the same thing as AM. TCP is a stream protocol, where connections exist. Here is his version of the TCP interface in nesc:

```
interface Tcp {
  command error_t bind(uint16_t port,
      uint8_t *buf, uint16_t bufsize);
  event bool accept( sockaddr_in6_t *to );
  command error_t connect( const
      sockaddr_in6_t *to, uint8_t *buf,
      uint16_t bufsize );
  event void connected();
  command error_t send( const void *buf,
      uint16_t len );
  event void acked();
  event uint16_t recv( void *buf, uint16_t
       len );
  command error_t close( bool force );
  event void closed();
}
```

Listing 3: TCP Interface

Here all of TCP's transitions are reflected as events. The only downfall to this version is that it requires changes to TinyOS itself, which creates significant overhead. The team originally intended to base the design off of this work but decided it would be easier to develop something on our own.

The next paper we read *IP is Dead, Long Live IP for Wireless Sensor Networks* [3], also by David E. Culler along with Jonathan W. Hui, contained more detail about implementing TCP in WSNs. The most relevant part of this paper to this work was its description of synchronous acks. According to Culler, for TCP to achieve the best efficiency, synchronous acks must be used in low-power mode where loss rates greater than 10% are common. This means the structure of an ack must be carefully considered.

This paper also discusses the costs of these systems. According to Culler the cost of transmitting a single byte is only 1.6uJ while the cost of transmitting an entire packet can be almost 630uJ, much higher. Culler also determined that despite all overhead costs the cost of listening for and acking messages is what really drove power consumption. He also discusses, albeit in retrospect, how in working on a resource constrained system, one should not sacrifice good software design and a layered approach.

Another reliable communication system worth mentioning is the spanning tree algorithm and protocol mentioned by Dr. Andrew Dalton in a paper on the DESAL language [2].

3. CONCLUSIONS

Overall this is a very interesting developing field with much new work to consider. Section 2.2 shows that a lot of the questions remained, even after developing this project. The majority of these questions were not addressed by other work in this field. This project raises several important questions about the development of reliable communication in WSNs.

By implementing TCP, UDP, and even IP on WSNs one can hope for a modicum of interoperability with existing devices, and this is possibly the logical conclusion of the project. In general it seems that most of the networking would be better off deeper in the kernel and network stack than the application layer, as it is now. Many believe neither TCP nor UDP really support WSNs and aside from a minimal implementation, we are inclined to agree. A new protocol should be developed specifically for the type of periodic transfers common to WSNs.

During the development of this program the team has become very familiar with nesC and all of its idiosyncrasies. We furthered our understanding of the problems of reliable communication not just in WSNs but in all aspects of networking. Addressing the questions asked in section 2.2 and fulfilling the implementation goals described in section 1 form the basis for future work.

4. ACKNOWLEDGMENTS

We would like to thank Dr. Dalton for these opportunities.

5. REFERENCES

[1] D. E. Culler. Wireless embedded inter-networking, June, 2008.

[2] A. R. Dalton, W. P. McCartney, K. G. Dastidar, J. O. Hallstrom, N. Sridhar, T. Herman, W. Leal, A. Arora, and M. G. Gouda. Desal alpha: An implementation of the dynamic embedded sensor-actuator language. In *ICCCN*, pages 541–547, 2008.

[3] J. W. Hui and D. E. Culler. Ip is dead, long live ip for wireless sensor networks. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, pages 15–28, New York, NY, USA, 2008. ACM.

[4] C. Ok, S. Lee, P. Mitra, and S. Kumara. Distributed routing in wireless sensor networks using energy welfare metric. *Inf. Sci.*, 180(9):1656–1670, 2010.