Reliability in Wireless Sensor Networks via a Modified TCP Implementation

Chris Blades
chrisdblades@gmail.com

Joseph Randall Hunt
randallhunt@gmail.com

Benjamin Rudolph
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 our particular implementation. The motivation behind this project was to create a reliable communications protocol for the `telosb` platform. This project attempts to solve the problems of reliable communication over radio 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 endusers 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 a reliable communication system has been an 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. [3] 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 several considerations must be made because of the platforms 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 create session management. We successfully implemented so-called ACKing but because of time constraints were unable to fully implement session management. We were also unable to do any high level empirical testing to determine the efficacy of our implementation. We mention several of the problems that held up development in the section 2.2

Paper Organization

Next this paper will discuss the implementation of TCP developed by our group, followed by an evaluation of this implementation. Finally we will discuss some related work.

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 our admittedly small implementation already improves on this. Traditionally TinyOS seems to operate on the Link layer as we can see in the message structure:

Listing 1: tos/types/message.h

```
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;
```


WCUWSN'10 Cullowhee, North Carolina USA

In this implementation of TCP for motes, the methodology 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.

Listing 2: TCPMessage.h

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

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.

The Test Application

The test application uses a simple, 14 byte long `SensorMessage` struct as the user type, though something much more complicated can be easily pushed in. 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 in this case) 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

Overall our work was very well done given the number of problems we ran into a long the way. Below are the problems and how they affected our 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 our group had more familiarity. These two aspects lead to a lot of difficulty in finding errors when they occurred. 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. Which illustrates another problem in our development.

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

Our group had a relative unfamiliarity with the language which lead to many time consuming errors that an experienced nesc developer would not have had to deal with.

Other topics in this particular area that would have been interesting to explore were the use of tasks and how this improved or degraded performance of our system.

2.3 Related Work

Of all the projects we have covered the one that works best with 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 though is a stream protocol, where connections exist. Here is his version of the TCP interface in nesc:

Listing 3: TCP Interface

```
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();
}
```

Here all of TCP's transitions are reflected as events. The only downfall to this version was that it required changes to TinyOS itself, and created significant overhead. We originally intended to base our 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* [2] also by David E. Culler along with Jonathan W. Hui, had some more detail about implementing TCP in WSNs. The most relevant part of this paper to our work was it's 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 and acking messages is what really drove power consumption. He also discusses, albeit in retrospect, how just because you are working on a resource constrained system does not mean you should forsake good software design and a layered approach.

3. CONCLUSIONS

Overall this is a very interesting and developing field with several interesting projects to consider. Given more time our team would have liked to answer several more questions about this particular project: How does the system behave in low power mode? At what distance does communication become completely unreliable? How standards compliant should one be? What parts of TCP should NOT be implemented? All of these questions were left unanswered or unaddressed by other work in this field. Overall what we found was that this project created far more questions than it answered.

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. Many believe neither TCP nor UDP really support WSNs and we are inclined to agree. A new protocol should be developed specifically for the type of periodic transfers common to WSNs.

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] 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.

[3] 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.