

# Securing Flooding Time Synchronization Protocol in Sensor Networks

**Jocelyn Adams**  
Engineering Physics  
Brown University  
[Jocelyn\\_Adams@Brown.edu](mailto:Jocelyn_Adams@Brown.edu)

Graduate Mentor: Tanya Roosta  
Research Supervisor: Dr. J. Mikael Eklund  
Faculty Mentor: Prof. S. Shankar Sastry and Prof. Ruzena Bajcsy

Friday, August 4, 2006



TRUST Summer Undergraduate Program in  
Engineering Research at Berkeley (SUPERB) 2006



Department of Electrical Engineering and Computer Sciences  
College of Engineering  
University of California, Berkeley

# Securing Flooding Time Synchronization Protocol in Sensor Networks

Jocelyn Adams

## Abstract

*Time Synchronization is essential for data transmission through wireless sensor networks (WSN). Each individual node in a WSN contains a crystal clock. Although the natural frequency of each crystal clock remains relatively constant, the quality of the crystal and environmental factors such as temperature cause the frequency to drift over time. In order to ensure that each node broadcasts the same global time within the micro-second range a time synchronization protocol must be implemented. Several protocols have been developed however none has been developed with security in mind. It is easy for an adversary to compromise a node causing faulty time sync messages to be sent out into the network.*

*The Flooding Time Synchronization Protocol (FTSP) is the proposed protocol. The FTSP is made up of several component modules written in nesC and is implemented in the TinyOS environment. TinyOS is an open source operating system developed specifically for network systems. In order to investigate the effects of injecting bad time synchronization messages into a WSN implemented with the FTSP, one or two nodes in a testbed of telos motes must be programmed to send faulty time sync messages. In order to determine the level security of the FTSP, the resulting global time errors will be recorded and evaluated.*

## 1 Introduction

As Ubiquitous Computing becomes more integrated with the environment the question of security arises. There have been numerous proposed applications of sensor networks many of which are security sensitive. These include surveillance, emergency disaster relief, battlefield intelligence gathering and health care for the elderly. These networks are made up of low-power micro electro-mechanical communicating computing systems embedded into the environment. Because it is imperative that these devices use minimal power consumption they transmit data via serial communication. In order for successful data transmission, the clocks of each mote must be properly synchronized. Because time synchronization is essential for successful communication between nodes in a wireless sensor network, were an adversary to compromise the time of a node it is likely that the applications being executed would fail. There have been several proposed time synchronization protocols developed specifically for wireless sensor networks. All have been designed to be compatible with these limited resource devices. None has been designed with security in mind.

## 2 Adversary Model

Because a wireless sensor network is made up of several devices communicating over a radio channel, each mote is vulnerable to physical capture by an adversary. We are attempting to model an inside attacker who has

access to encryption keys and code used by the network. The modeled attack will be one in which false time synchronization messages are injected into a network and accepted by other nodes in the network without being detected as compromised information. Other methods have been implemented in order to ensure security of the time synchronization protocol, but fall backs of these methods include aborting the time synchronization process or added data redundancy through pairwise node authentication. The objective of this investigation is to eventually filter out bad data from compromised nodes eliminating the need to either abort the time synchronization process or add data redundancy. [2]

### 3 Clock Drift

Each node basis its notion of time on the natural frequency of a crystal quartz clock. Ideally, real time could easily be constructed given the frequency of the crystal. Real time could be expressed as stated below, where  $\omega(\tau)$  is the frequency of the crystal and  $k$  is a constant.

[2]

Although the frequency of oscillation remains relatively constant, slight changes such as temperature, pressure and voltage cause this frequency to fluctuate. In order to compensate for this clock drift, a time synchronization protocol must be implemented.

### 4 The Flooding Time Synchronization Protocol

The Flooding Time Synchronization Protocol (FTSP) is the proposed time synchronization protocol. FTSP is made up of several component modules written in nesC and is implemented in the TinyOS environment. TinyOS is an open source operating system developed specifically for network systems. This protocol synchronizes the local time of a root node to possibly several receivers by broadcasting a single radio message timestamped at both the sending and receiving end.

FTSP provides multi-hop time synchronization. In the case of a multi-hop network each node receives a message, updates its globalTime, and broadcasts its updated globalTime into the network.

Each packet includes the following information:

- rootID: the mote ID of the node sending the TimeSyncMsg
- nodeID: the mote ID of the node receiving the message
- seqNum: the sequence number of the root
  - with each message sent out from the root a TimeStamp is stored in an array. The sequence number gives the place the current TimeStamp is being stored in this array.
- sendingTime: the time the message was sent in the global time of the sending node
- arrivalTime: the time the message was received in the local time of the receiving node

If the sequence number of the received message is older than the previously processed message it is ignored. If not, the node updates its local time and converts that time to global time:

Conversion of from localTime to globalTime:

$$\text{globalTime} = \text{localTime} + \text{offsetAverage} + \text{skew}(\text{localTime} + \text{localAverage})$$

Updated arrivalTime (localTime of the receiver) in terms of the sendingTime (globalTime):

$$\text{localTime} = \text{globalTime} - \text{offsetAverage} + \text{skew}(\text{globalTime} - \text{offsetAverage} - \text{localAverage})$$

Where localAverage is the average of the most recently stored localTimes (due to limited computational and memory resources only small number of entries can be stored and used) and the offsetAverage is the average offset of the local time to global time. Skew is defined here as the remainder of  $(\text{skew} - 1)$ .

It may be worth noting that in the above calculation of localTime, localTime is estimated to be  $(\text{globalTime} - \text{offsetAverage})$  in order to simplify the equation. This assumes that the globalTime and localTime are comparable. This assumption will increase the effectiveness of the aforementioned adversary model.

The skew is calculated using a least squares linear regression:

$$\text{skew} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

$x_i$  : localTime  $\bar{x}$  : localAverage  $y_i$  : offset  $\bar{y}$  : average of offset [1]

## 5 The Compromised Node

In order to insert a compromised node into the network, a telos mote must first be programmed to inject bad timesync messages. This can be done in the module TimeSyncM.nc of the application TestTimeSync. In order to avoid complications with the code, the error must be inserted into the code right before the node's time is broadcasted into the network. Inserting a localTime right before the line:

```
outgoingMsg->sendingTime = globalTime - localTime;
```

In the method `sendmsg()` will cause the outgoing timesync message to be wrongly offset by however much you assign localTime to be. The application is built to reject broken nodes and does throw nodes out of the network if the offset between the incoming and stored messages is greater than the *ENTRY\_THROWOUT\_LIMIT*.

*For Telos Motes : ENTRY\_THROWOUT\_LIMIT = 800*

If the localTime is changed by more than 800 the compromised messages will be rejected by the receiving nodes.

The current corrupted file looks as follows:

```
localTime = localTime + 700;    outgoingMsg->sendingTime = globalTime - localTime;
```

## 6 Parsing Data Packets

A parsing application (ParseDiagMsgs.java) has been developed for TestTimeSync in the Mica platforms, but no such application had been developed for the Telos platform. To examine the incoming packets I used the

application Listen.java. This application displays the packet as a string of hexadecimal numbers. An example is below and I have segmented it out.

Picture goes here....

Both globalTime and localTime are 32 bits. In order to decode the globalTime and localTime of messages received at the base station perform the following operations:

Example: 9B E4 FB 07

Time in seconds =  $(hex2dec(9B) + hex2dec(E4) * 2^8 + hex2dec(FB) * 2^{16} + hex2dec(07) * 2^{24}) / 32678$

The parsing application will output the motelID of the sending mote as well as the globalTime and localTime stored in the packet.

## 7 Future Work

The goal is to set up several motes implemented to run TestTimeSync in a multi-hop network, insert one or two motes with false localTime, and evaluate the results. Because, at full power, the range of Telos full has been measured to be up to 30 meters, the motes should be programmed to run on the lowest power. This will allow for the experiment be run indoors. In order to reset the power of the motes you can use the CC2420ControlM.nc module in the CC2420Radio directory and your Cygwin window as an interface. The command is SetRFPower and the lowest power is 3 which refers to -25dbm.

The developed parsing program and testbed will serve to test any type of corrupted message injection into the network. It would be interesting to investigate the effects of overflowing the network with messages. There are many types of invasion that can be modeled.

## 8 Acknowledgement

### References

- [1] D. Cox. Time synchronization for zigbee networks. Technical report, University of Alabama in Huntsville, Dec. 2006.
- [2] T. Roosta and S. Sastry. Securing flooding time synchronization protocol in sensor networks. Technical report, University of California, Berkeley, 2007.