Distributed Computation of Average Temperature within Wireless Sensor Networks

Sameer Lal
Columbia University

1 Introduction

1.1 Motivation

Due to various constraints in power and computational ability of sensors, Wireless Sensor Networks (WSNs) have maintained a specific “centralized” architecture. In this architecture a sink node, or central hub, collects data from all nodes in the network via one hop or through multiple hop communication through other nodes. In many scenarios, as shown in Figure 1, this central or sink nodes communicates with a more powerful source of computing capability (traditionally a computer, but recently the cloud) to process the data and then actuate on the findings. Though stable and highly used, the “centralized” architecture has significant drawbacks surrounding latency, security, and energy consumption that have made it difficult for WSN to truly react to environments and be capable of executing 21st century WSN applications. Issues with latency arise usually in the communication between WSN and external computing power. Failure of data transfer in these bottleneck parts of the architecture can impact robustness, as well as provide a security weakness. Most importantly, the collection of the data and its transfer to an external source tends to encroach issues of privacy.

1.2 Problem Framework

Recently, improvements in WSN technology such as computing power, energy consumption, and transmission speed have made the move to a more improved model more feasible. Sensors networks now have the ability to run slightly complex algorithms and perform significant amounts of calculation. With this in mind, we looked to provide a new architecture for WSNs that could overcome the drawbacks of the “centralized” model. We proposed to move computation within the network, removing all external sources of computation. Since WSNs have limited computational power, any processing would have to be performed in a distributed manner in order to be feasible with scaled up network sizes. In addition to improving upon the “centralized” WSN architecture in latency, robustness, and security, the goal was to keep essential functionality and features such as ease of setup or the ability for nodes to drop out or be added to the WSN. As a measure of quality the new architecture would be benchmarked against the centralized model for improvement when calculating the average temperature of all motes in a WSN.

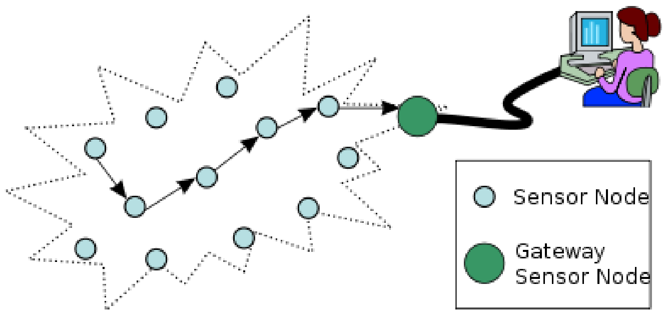

Figure 1:An example of the "centralized" architecture (Wiki)

1.3 Current Solutions

Though extensive work has been done on the potential applications and computations WSNs can do, little work has been done regarding in network computation. There have been a few successful attempts at computation of average sensor readings in a network. Namely, TinyDB (see reference 7) is a framework which enables for the WSN to be queried from an external point (like a database) for certain aggregates including the average of all readings. It is unclear exactly how this aggregation is processed, whether it happens on a central node or whether it is done in a distributed fashion, but this calculation is certainly done without any source of external computing power. Others have come up with theoretical means by which to group nodes and provide a platform for distributed computation, but no real world tested solutions seem to have been developed.

1.4 Testing and Implementation Framework

The new architecture or framework for distributed computation of average temperature was built in TinyOS, an operating system written in NesC (a C variant) for multiple research based sensors. The sensors in testing were Telosb Motes developed by Crossbow Technology which have multiple types of sensors including temperature sensors. Additionally, the Cooja simulator was used to simulate larger scale networks. In designing our network architecture, our key assumption was that all motes in the network could be started and run synchronously such that their internal timers or clocks would be all running parallel in time.

2 Theoretical Analysis

2.1 Network Topology

The first step to design an effective architecture for distributed computation of average temperature was to analyze the most effective network structure or topology. Based on their drawbacks, some common network protocols were discarded as potential variants. The Star, Bus, Ring, and Line topologies, though simple and would have ease of setup, had many drawbacks that were associated with the centralized architecture. In these architectures, processes would not be concurrently distributed and rather would aggregate in size and complexity as data moved through the network, potentially leading to significant latency when scaling the network in size.

Additionally, these networks sacrificed simplicity and ease of setup for robustness. In a network with a Star or Ring topology, the removal or attack on one node could bring the entire network down or surrender the processed data.

Other network topologies such as Mesh or Fully Connected were slightly more robust and had potential means of overcoming the aggregation of computation by running some processes in parallel. The obvious problem with a Fully Connected network topology is that realistically very few WSN could have all motes connected to each other. A Mesh network topology solves this problem, yet due to the undirected nature of the topology, algorithmically running concurrent processes would require significant work in setting up the network, a drawback that would render the topology suboptimal.

2.2 Binary Topology

Tree network topologies, specifically a binary tree topology, theoretically was found to beoptimal for the described functionality. In terms of latency and the distribution of computation, a binary tree topology causes the most concurrent and evenly spread processes of calculation. In a perfect binary tree each leaf node will have *log(n)* steps of processing with each leaf node initially having *log(n)* concurrent processes. Additionally, each parent node has an understanding of how many children it has (exactly 2), which would be helpful in the case of dropping or adding children. Setup could be done using a slew of algorithms and metrics for mote values with relatively simple computation. The only caveat would be to build a protocol for the rebalancing of a tree if and when motes were dropped or added to the network. A binary tree network topology theoretically satisfied the functionality of the original “centralized” architecture while improving on the emphasized principles.

2.3 WSN Characteristics and Network Requirements

Although a binary tree network topology was theoretically sound, it still lacked the feasibility for implementation in most common WSNs. Sensor technology and thus a WSN is usually very dynamic. Successful packet transmissions and link quality are variable even over the course of a few seconds. Even small changes in the environment that are common, such as the movement of people or objects, could in fact cause established links to fail and packets to be dropped. Motes therefore commonly will drop out of the network momentarily and then re-enter with different link qualities even though the motes are stationary. In the context of a binary tree topology, this would require the network to expend a lot of energy and computational effort rebalancing. Conspicuously inefficient and suboptimal, a more robust and properly tested (in a real WSN setting) network topology was needed.

2.4 CTP

One of the most robust and well tested network protocols is the Collection Tree Protocol (CTP). Developed in 2009, the aim of the protocol is to guarantee optimal robustness and transmission of packets through the network. The protocol establishes a root of the tree (given as an input) and creates a tree using a specifically designed metric of link quality known as ETX. ETX is a metric for expected

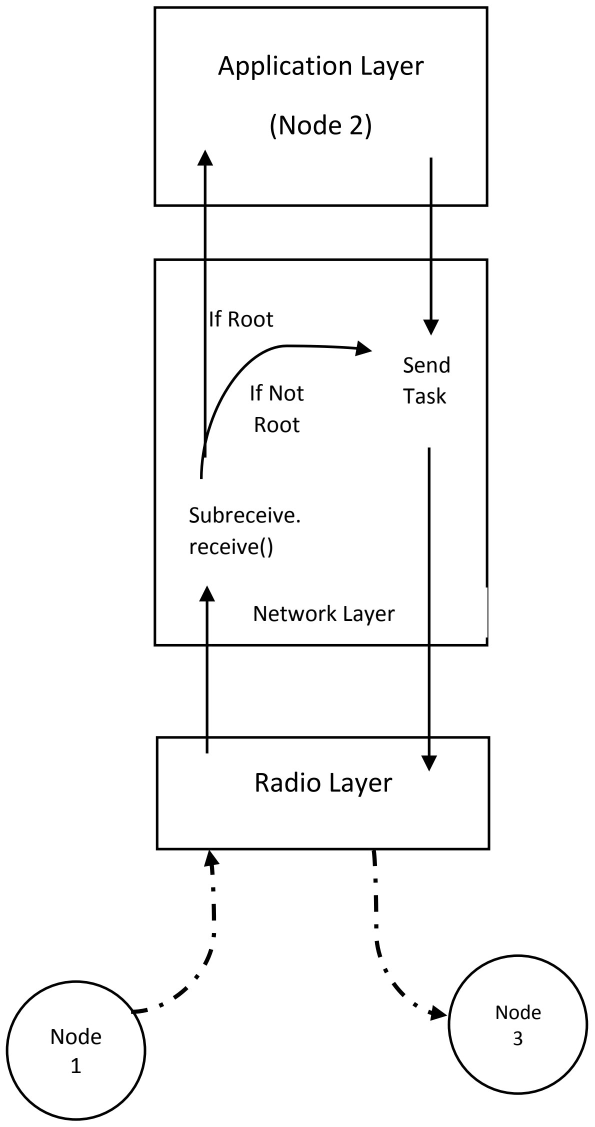

Figure 2: CTP Forwarding on Fennec Framework Layers

transmission count. Essentially, a ratio for the number of packets required to be sent for a successful transmission between two motes, thereby establishing the link quality. Nodes in network will use a variant of Dijkstra's algorithm with ETX values signifying the cost to build the most robust path, or that with the highest link qualities, to the root of the tree. Given this structure, CTP maximizes the chances that data at any non-root nodes will successfully reach the root. CTP has become an industry standard and is widely used, with functionality to deal with variation and perturbations in the network due to changes in the environment. Therefore it was logical to build on or modify CTP to accomplish the goal of robust in network distributed computation.

3 Implementation

3.1 Fennec Fox

Implementation was done in NesC for the TinyOS operating system which Telosb sensor motes run. In order to facilitate development, the Fennec Fox framework was leveraged. Fennec Fox is a middleware framework that allows for the compartmentalization and abstraction of WSN functions into three distinct layers: Application, Network, and Radio. This enabled the computation protocol and network protocol for distributed computation of the average temperature to be developed in separate modules. One of the prominent features of Fennec Fox is the ability to switch different modules for these different layers in order to change functionality without having to re-develop large parts of code. The new architecture was designed such that different types of computation (min, max, standard deviation, etc.) could be developed as individual modules that could be switched in place of the module for average calculation. With this in mind, all computation was developed as an application layer module. There existed a Fennec module that implemented the CTP, and this code was modified in order to accomplish the task of distributed computation.

3.2 CTP Compute Module

The CTP is intended to be a data collection process. As nodes collect or generate data, the protocol forwards packets towards the root. As shown in Figure 2, packets received from previous hop nodes go through a check of whether the node is in fact the root, if not, the packet is sent to the next hop in the network. If the node is the root, packets are pushed to the application layer where it can be processed. The developed Fennec module, *ctp_compute*, takes in a single parameter, the root node of the network. The modification made to CTP is changing when packets are forwarded and when they are sent to the application layer. In *ctp_compute*, all packets are forwarded to the application layer for calculation.

All other functionality of CTP for network setup and maintenance remained unmodified. The network would have the same topology and functionality as the original CTP except packets are forwarded to the next hop and then processed rather than forwarded immediately towards the root.

3.3 CalcTempMetrics Module

In order to perform the computation of calculating the average temperature an application layer Fennec module, *CalcTempMetrics*, was developed. Although the computation of average is relatively simple, much of the difficulty was timing events in order for synchronous processing and reduced latency. The algorithm takes two inputs: the root node (also provided for the *ctp_compute* module) and the approximate number of hops in the network. The second parameter is provided to give the network a sense of how much cushion should be given to collecting data from child nodes. With more hops, the probability of failures increases and thus the network must give an increased margin slower resoponses to child nodes.

The computation, described in Figure 3 in the form of a finite state machine, can be seen as several distinct process loops. The overall process is started by a timer (denoted as Timer1 in Figure 3). This timer triggers all leaf nodes to push their temperature measurements to their parent (the next hop). The parents run a distinct process for the aggregation of child measurements. Upon receiving the first measurement a timer (denoted as Timer2 in Figure 3) is started. This timer's length is determined in milliseconds by:

$$T\ (milliseconds) = L * H * 150$$
$$where\ H\ is\ the\ input\ number\ of\ Hops, and\ L\ is\ the\ last\ number\ of\ children\ from\ which\ a\ measurement\ was\ received.$$

For the length of the timer, the other child nodes have the opportunity to send their readings to the parent node. Once the timer has fired, the local average is calculated and pushed to the parent node. By having timers of variable lengths, non-leaf nodes allow for the switching of child nodes to other parents.

The root node has a longer central timer (Timer1 in Figure 3) to allow for highly imbalanced branches of the collection tree to still be counted in the calculation of the overall average. Once this timer fires, all the reading values are summed and then averaged over the number of readings.

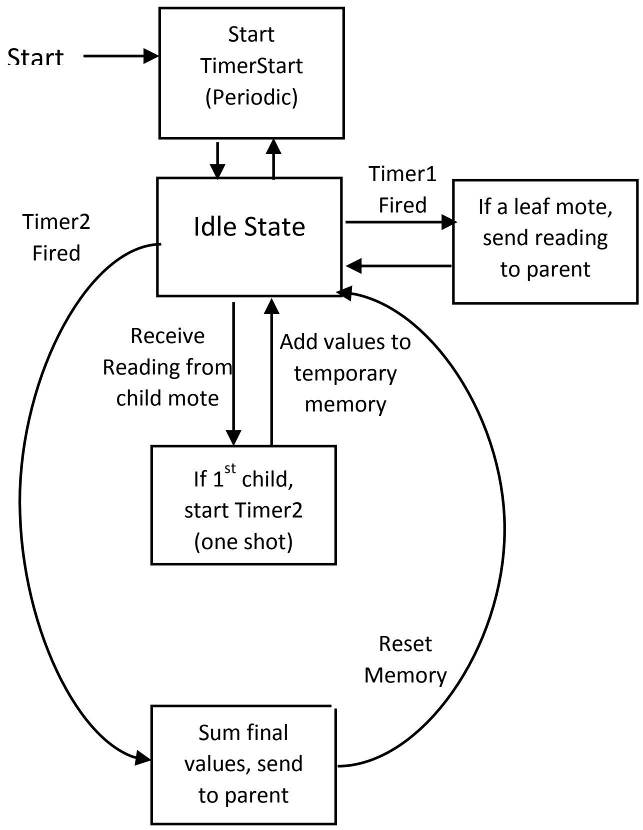
Start
Start TimerStart (Periodic)
Idle State
Timer1 Fired
If a leaf mote, send reading to parent
Timer2 Fired
Receive Reading from child mote
Add values to temporary memory
If 1st child, start Timer2 (one shot)
Reset Memory
Sum final values, send to parent

Figure 3: An FSM of the CalcTempMetrics Module

4 Results

4.1 Cooja Simulation

The implemented modules for in network distributed computation of average temperature were tested on the Cooja simulator (part of the Contiki operating system). Simulations of networks with few nodes ran properly, able to calculate simple averages quickly within 15-20 ms. The issue that arose quickly was the asynchronous nature of the clocks on motes. This coupled with failed transmissions caused the state of the network to move into loops that resulted in inaccurate calculations of the average. These loops, such as triggering calculation protocols out of sync or sending measurements after calculations were processed, were eased by increasing the parameter values but as the WSN quickly scaled, especially with more hops these issues grew in severity.

What became apparent after several simulations is that although such an architecture could work and it would not be necessarily slower than the "centralized" architecture, our main assumption of clock synchrony was highly inefficient. Having parameters to deal with this may have alleviated the problem, but from the logs of the simulation it became clear that the timing of these clocks greatly affected the accuracy and a different means of acting synchronously should be used.

5 Conclusion

Ultimately, the goal to develop a new architecture to enable in network distributed computation was accomplished. The architecture was developed via two new Fennec modules that were tested in the Cooja simulator. Simulations confirmed that in network computation of the average was done with accuracy for small sized networks. The results of the simulator also displayed much room for the improvement of the computational and network processes. Moving forward, we hope to reduce the use of mote timers and provide more accuracy in calculations. Once a more mature implementation has been developed it should be deployed and tested on real sensor test beds for a true measure of robustness.

References

[1] Gnawali, Omprakash; Fonseca, Rodrigo; Jamieson, Kyle; Moss, David; Levis, Philip (2009). "Collection tree protocol". *SenSys*: 1–14

[2] Fonseca, Rodrigo; Gnawali, Omprakash; Jamieson, Kyle; Kim, Sukun; Levis, Philip; Woo, Alec (2006–2007). "CTP". *tinyOS*

[3] De Couto, Douglas S. J. (2008) "High-Throughput Routing for Multi-Hop Wireless Networks".

[4] Philip Levis, Neil Patel, David Culler, and Scott Shenker. (2004) “Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks.” *First USENIX/ACM Symposium on NSDI.*

[5] Eugene Wu, Samuel Madden. (2013) “Scorpion: Explaining Away Outliers in Aggregate Queries”.

[6]Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. (2002) “TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks”. *Proceedings of OSDI.*

[7] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. (2005) “TinyDB: An Acquisitional Query Processing System for Sensor Networks”. *TODS.*

[8] Nildo Ribeiro Junior, Marcos A. M. Vieira, Luiz F. M. Vieira, Omprakash Gnawali. (2014) “CodeDrip: Data Dissemination Protocol with Network Coding for Wireless Sensor Networks”. *EWSN.*

[9] Kaisen Lin and Philip Levis. (2008) “Data Discovery and Dissemination with DIP.” *IPSN*

[10] Jen-Yeu Chen, Gopal Pandurangan, Dongyan Xu, "Robust Computation of Aggregates in Wireless Sensor Networks: Distributed Randomized Algorithms and Analysis", *IEEE Transactions on Parallel and Distributed Systems*, 17(9), 2006.

[11] Ahmed, K., Gregory, M. (2011) “Integrating Wireless Sensor Networks with Cloud Computing”, Seventh International Conference on Mobile Ad-hoc and Sensor Networks (MSN), Beijing, China, 16-18 Dec. 2011, pp. 364-366.

[12] Marcin Szczodrak, Yong Yang, Dave Cavalcanti and Luca P. Carloni (2013) “An Open Framework to Deploy Heterogeneous Wireless Testbed for Cyber-Physical Systems”

[13] Muhammad Hamad Alizai, Olaf Landsiedel, Jo´ Agila Bitsch Link, Stefan G´otz, Klaus Wehrle. (2009) “Bursty Traffic over Bursty Links”.