Wireless Ad-hoc and Sensor Networks
COSC 418
Brief Project Description

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury
email: andreas.willig@canterbury.ac.nz

August 22, 2011

---

1 Project Summary

The project centers around a modification of an existing protocol for data collection. More specifically, the CTP protocol [1], [2] establishes a tree that is rooted in a special node, the **sink node**, and whichs other nodes are normal sensor nodes sending their data to the root along the tree. More specifically, a sensor node maintains a **parent node** that is either the sink or, if the sink is not directly reachable, is another sensor node that is "closer" to the sink node. The precise meaning of the word "closer" here is related to a certain cost model: a node always selects the neighbor that has the closest cost to reach the sink. The costs from a node to the sink is the sum of two quantities:

- The direct transmission cost between the node and its parent.
- The cost of the parent, i.e. the total cost of sending the packet from the parent to the sink, determined recursively.

In CTP the tree is organized so as to minimize the expected total number of transmissions (including retransmissions) that any node experiences. To achieve this, CTP relies on the underlying link layer to supply the `ETX` (expected transmission number) value of an individual link, which roughly counts the expected number of trials needed until a packet is successfully transmitted over a link. The CTP protocol so far uses only these `ETX` values as its cost criterion.

The **goal** of the project is to extend CTP to also include a load-balancing component, i.e. an approach by which nodes that have been working very hard in data forwarding

get a break. You need to design and implement such an extension. Furthermore, you are asked to design experiments which prove the efficacy of your approach.

The main work on the project is expected to take place during the term break and the second term of the semester. The first lecture slot of the second term will again be devoted to the project, there will be room for questions and answers and you will have to give a brief (five minutes) interim report about your project. During the last lecture of the second term you will present your project results to your peer students. This presentation should take 15 minutes. You should explain your protocol design and present your measurement results.

The marks for the project will be based on the interim report(10%), the achieved functionality of your code (60%), and the final presentation (30%).

2 Details

2.1 Preparations

- Read the TEP document and chapters one to seven of [3]. You should especially look at Section 6.4 and Appendix A.2 to learn about the usage of collection protocols (including CTP) in TinyOS applications.
- Read and understand the CTP implementation, which you find under

  `/opt/tinyos-2.1.1/tos/lib/net/ctp/`

  You should particularly look at the packet formats (see `Ctp.h`)
- The TinyOS source code in the virtual machine can be found under `/opt/tinyos-2.1.1/`. This is the root directory of TinyOS and contains a number of sub-directories. One of them, the `apps` sub-directory contains a number of example applications.
- You could use the `apps/MultihopOscilloscope` application as a starting point for your project, since it pretty much covers most of your needs, and extending this one is simpler than writing an application from scratch. Create your own clone / copy of this application in your home directory and work from there. You should also inspect the `Makefile` of the `MultihopOscilloscope` application to see how the CTP protocol is included into the application and how you can instead use your own instance / copy of CTP (see next).
- Next you should also copy / clone the implementation of the CTP protocol, which can be found under `tos/lib/net/ctp` in the TinyOS source tree.
- As the very first thing make sure that your clone of `MultihopOscilloscope` uses your clone of CTP when compiling (i.e. modify the `Makefile` appropriately). Set up and run initial tests, you could for example use the LEDs on the nodes to visualize various activities. In those tests you should make sure that node identifiers are assigned properly.

2.2 Some Hints for Protocol Design and Development

Basically, the ultimate goal of your project is to come up with a new decision function for choosing a parent. Roughly speaking, right now a sensor node s chooses its parent as

$$p^* = \underset{p \in \{\text{direct neighbors}\}}{\arg\min} \; \text{ETX}_{s,p} + \text{ETX}_p$$

where $\text{ETX}_{s,p}$ is the ETX value for the direct link between the node s and its neighbor p (this is obtained from the local link layer) and ETX_p is the cost value that neighbor p has advertised as his (p's) cost to reach the sink.

This decision scheme needs to be extended so that s also incorporates information that it has about the load of its direct neighbors. If we denote the load information about a neighbor p available at node s as L_p^s then the new decision function could have the form:

$$p^* = \underset{p \in \{\text{direct neighbors}\}}{\arg\min} \; \left[\alpha \cdot (\text{ETX}_{s,p} + \text{ETX}_p) + \beta \cdot \text{L}_p^s\right]$$

where $\alpha \geq 0$ and $\beta \geq 0$ are configurable parameters determining the relative weight of the total ETX costs and the load of a neighbor in the decision. Please note that all the information that the CTP protocol is concerned about so far (notably the ETX_p values) are used in the same way as before, so these aspects of CTP do not need to be changed at all.

The first thing you have to decide in your protocol design is the precise meaning (and subsequently: the representation) of the values L_p^s. I will outline some of the possible design options, there are likely more:

- In the first design option no explicit communication about the L_p^s values takes place, instead node s estimates them purely locally. Perhaps the simplest such scheme would be that s counts the number n_p^s of data packets that s *itself* has sent to p so far and simply set $\text{L}_p^s = n_p^s$ or $\text{L}_p^s = \phi(n_p^s)$, where $\phi(\cdot)$ is a non-negative and monotonically increasing function. In a more involved scheme node s would attempt to overhear all data packets that p sends to p's parent and use this overheard information to estimate p's load. **Question:** What are the benefits and drawbacks of either approach?

- In the second design option a neighbor p estimates its own load L_p and communicates this estimate to node s (which then clearly uses $\text{L}_p^s = L_p$), or perhaps broadcasts this value to all of its neighbors. In this class of approaches you need to carefully consider a representation for L_p (How many bits to use? How are the values encoded?). Furthermore, you need to decide whether the L_p values just reflect the *local* load at node p or whether L_p reflects the load *on the entire path* from p to the sink (similar to the ETX_p values). There is a further issue in such a communication-based scheme: when a neighbor p broadcasts a new value L_p, he might trigger a new decision at *all* of its neighbors. **Questions**: why is this potentially a problem? Do you want to do something against it? And if so: what?

In your protocol development you should proceed in small steps and test each step thoroughly before making the next step. For example, if you decide to let p communicate its load L_p, then you could make the following small steps:

- Decide on the number of bits to use for L_p.
- Extend the packet formats for CTP by a header field suitable for carrying L_p values. Modify the `struct`'s in `Ctp.h`, recompile everything and check that your application still works.
- Next add the logic by which a node p calculates its local load estimates L_p and let p write those values into the CTP packets / headers whenever a new packet is sent. Recompile and check.
- Next include the values received from p into s's decision function. Recompile and check.

You should get the idea. To say it again: proceed in small steps, make sure every step works before taking the next one. And above all: **Use a version control tool, e.g. CVS!**.

2.3 Demonstration

You are expected to demonstrate the efficacy of your design by means of an experiment. More specifically, you should use a network scenario like the one depicted in Figure 1, in which there is one sink node and three sensor nodes. The sensor nodes should be arranged such that:

- Sensor nodes 1 and 2 are direct neighbors of the sink
- Sensor node 3 is a direct neighbor of both sensor nodes 1 and 2, but not of the sink.

The easiest method to achieve this is by varying the distances between the nodes appropriately. To keep the distances small, you should use a small transmit power for all nodes.

You should design your demonstration / experiment such that it convinces me about the proper operation of your protocol. You could either use the LEDs of the nodes to visualize states or actions of your scheme, or you could extend the protocol to include "debug data" into the user packets.

References

[1] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo. The collection tree protocol (ctp). TEP 123, TinyOS Network Working Group, August 2006.

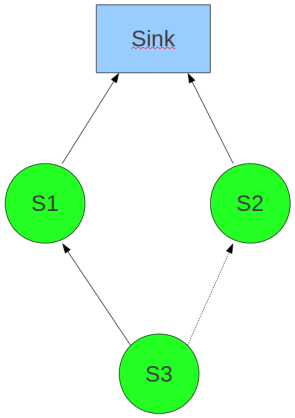

Figure 1: Example scenario for demonstration

[2] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, David Moss, and Philip Levis. Collection tree protocol. Technical Report SING-09-01, Stanford Information Networks Group, Stanford University, 2009.

[3] Philip Levis and David Gay. *TinyOS Programming*. Cambridge University Press, Cambridge, UK, 2009.