Matt Buland
February 1st, 2014
Homework #3.C
Time: 2.5 hrs

Summary

This paper summarizes the lessons that a reasearch team learned when doing a few of their deployments. The suggested practices focus on creating a robust system that behaves with as few surprises as possible, which includes some things to look for when testing, how to create an accurate testing environment, and some tips to ease the pain of inevitable failures.

Main Areas

Data integrity, efficency matters, and keep in small and simple are the main areas of a successful deployment. Data integrity (Data You Can Trust) is the main goal of the deployment: to gather good data that researchers can use. By developing good calibrations for the sensors, and typical behaviors about the expected data, faults can be easily identified. Knowing when something is wrong with the deployed sensors is extremely important for outdoor deployments, since theres usually many scenarios that would require intervention: such as most motes being destroyed, moved, dieing, etc. Though intervention isnt always a possibility, the baseline calibration will always be important in determining the quality of data, and has a direct effect on research outputs. Efficiency matters is a section on the importance of having an easily accessed test environment that can be used for practice and situation prediction. Being able to test a network of motes as quickly as possible has a direct effect on development velocity, and also allows for easier identification of future problems. Keeping it small and simple (KISS) is a common development technique that tries to highlight any problems that occur. With a simple codebase, bugs are usually easier to see, fix, and prevent. With this also comes some fault tolerance; the simplicity causes most possibilities to be directly addressed, so that the network is prepared for anything.

SW/HW Development

One thing I wouldnt have thought of is version stamping packets. By doing this, coupled with source control, the code used to process a particular piece of data can traced for error. Of certain importance is the physical packaging around the motes. To properly prepare for the vicious outdoors, lots of care and planning must go in to protecting the fragile electronics and sensors. If a mote breaks or gets damaged, it can basically be useless, potentially producing wild results. Time drift is also a really important aspect to account for. Even without any crazy environment changes, theres usually time drift to account for; adding in even slight temperature changes can vastly change the drift, so prepare for it test for it, and try not to depend on it too much, just in case it fails.

Testing and deployment

This semester, I certainly would not have thought of serious time drift. Test the environment for factors that would affect this. Also of importance, practice physically deploying; the better the team is at this, the less could go wrong. Though inside the scope of this class, any deployments we do will likely be test deployments. Still if we did one later, this would help in that. In general, the more testing and the better testing thats done, the better the actual deployment should be.

Deployment

In actual deployment, there are often few chances for changes, which is why preparation is stressed so much. When active, keep a careful eye on everthing possible: the data, the weather, and to the best of your ability, the motes. There may be situations, however, where you need to sit back, and let things get worked out. Also, keep multiple backups of all the collected data; we dont want to lose any of it. All the collected data has SOME sort of use, such as tracing the system, and determining faults; maybe even correcting some. Once the deployment is over, its good to do a final calibration, check hardware, and disassemble everything, so it doesnt conflict with any other tests or projects.

Wonders

I wonder if there are good ways to correct for time drift. Are there other factors that can affect the crystals?

Why does everone seem to be reinventing the network stack? Sure, TCP, IP, and MAC are a little verbose for motes, but hasn't there been some sort of common implementation than everyone can use? Does tinyOS implement one? Did it then, versus now?

Issues

I didnt have too many issues with the paper; since its an alacarte style paper, you can kinda ignore the parts that cant apply to your specific area.