Motivations Behind SOS

Roy Shea, Simon Han, Ram Rengaswamy

February 20, 2004

Abstract

SOS. For those who are seeking the sunny side of the street we have 'Save Our Sensors'. Individuals seeking a more fundamental approach to life can use 'Sensor Operating System'. Mathematicians may find amusement in the recursion present in 'SOS Operating System'. Or choose your own. The possibilities are almost endless. For now, it is simply SOS.

1 Introduction

SOS is a new operating system for sensor networks that attempts to remedy some of the limitations resulting from the static nature of many of its precursors. SOS implements a messaging system to sever ties between the core operating system and individual applications. Individual applications, referred to as modules, can be loaded or removed at run time without interrupting the core SOS operating system.

Modules are written in the C programing language and compiled using standard C compilers. Rather than using a main function, modules implement a basic message handler, in the form of a single switch / case block, that directs messages to their module specific code. This programing interface makes it easy for programmers to begin working with sensor networks, without having to learn custom languages. Support for compiling unmodified TinyOS modules directly into SOS application code is very important to SOS, and will be discussed in more detail below.

2 Reconfigurability

Reconfigurability is the primary motivation and goal for SOS. For the domain of sensor networks, we define reconfigurability as the ability to modify the software on individual nodes of a sensor network after the network has been deployed and initialized. This provides the ability to incrementally update the sensor network after it is deployed, Reconfigurability also makes it possible to add new software modules to a sensor network after deployment, and remove unused modules when they are no longer needed. The growing tensions between large hard to update networks and complex applications with incremental patches has made reconfigurability an issue that can no longer be ignored.

Reconfigurability opens the door to three areas neglected in much current work in sensor networks: fault tolerance, heterogeneous deployments, and new programing paradigms.

2.1 Fault Tolerance

The ability to dynamically add or remove modules enables the construction of more fault tolerant software. Two examples of improved fault tolerance follow. The first gain comes through the ability to incrementally deploy software in the sensor networks. This allows problematic modules to be updated with newer and more stable versions, without needing to physically access the

sensor network. The second gain depends on the ability to remove modules from the sensor node. In particular, troublesome modules can be identified and removed from a node with minimal interruption of node services.

2.2 Heterogeneous Deployment

Heterogeneous deployment is an area of sensor networking that is beginning to receive more attention. Already, a number of groups have identified the benefits of deploying heterogeneous types of nodes. Often this takes the form of sensor clouds made up of simple motes, connected by a more powerful backbone network made up of XScale class devices. An area of heterogeneity that has been mostly ignored is the deployment of heterogeneous applications on top of homogeneous nodes. Focusing on mote class devices, this heterogeneity could enable more specialized applications to be built and configured, without worrying about the overhead or interactions of other applications. Reconfigurability in SOS enables the deployment of such a system by loading different modules onto nodes after deployment.

2.3 New Programing Methodologies

Very few methodologies for application development for sensor networks exist. Reconfigurability opens the door to new programing paradigms. Before reconfigurability, most applications were built as a single monolithic kernel. Development was eased by dividing this monolithic kernel into easier to understand components that were combined during compilation. SOS takes this a step further by providing the components separate through the compilation phase. This enables construction of true software agents and active networking. Finally this flexibility may also lead to advances in macroprogramming, since users no longer focus on programing at the node level (monolithic kernel) but at the application level.

3 SOS State and Goals

SOS is currently up and running on Berkeley Mica2 motes. The code base supports the basic primitives used in sensor networks including: UART, timers, ADC, physical layer and MAC for radio, and limited access to sensors on the sensor board. Specific to SOS we have the ability to: add / reload / remove modules, basic watch dog timer, simple memory manager, and application development. Two applications are being developed for SOS. The first is based on Visa and uses a distributed Voronoi spatial analysis to calculate the area that individual nodes are responsible for and if a specific sensor can cove this area. A second program uses motes for augmented recording.

3.1 Short Term Goals

Over the next four weeks we hope to implement an RPC mechanism for use in SOS. This will provide a protocol with which to compare other work in SOS. We also plan to formally outline the algorithms used for reprogramming the sensor network. This includes flooding algorithms, point programing, and module discovery. Finally, we plan to continue developing a more robust kernel. This include implementing memory integrity checks and initial work on discovery of misbehaving modules.

3.2 Medium Term Goals

SOS is scheduled to be fully up and functional by the end of June, 2004. This includes a mature implementation of a watch dog timer that helps to detect and evict misbehaving modules. A PC side debugging solution will be implemented. This will provide increased support for examining and debugging applications on a PC before running on motes. Module dynamics will continue to be refined throughout this period of time. A larger application base will be developed by this time.

A primary component to this base will be from TinyOS. In particular, we will produce a compiler that translates unmodified TinyOS code into into SOS modules. This will allow full backwards compatibility with TinyOS applications while providing the benefits of modular reconfigurability.

We will use two or three new SOS applications to drive this development and refine interfaces. One such driving application will use SOS to provide person aware lighting within the NESL lab.

3.3 Long Term

By summer of 2004 SOS will support more advanced application development and real deployments. Over the summer months, work with SOS will transition from operating system development to platform use. We hope to begin working with external groups to use SOS in deployed sensor networks.

Research will also use SOS as a foundation platform. Topics to be examined will include: the role of agents and messaging in sensor networks, challenges in distributed control within sensor networks, language issues with sensor networks, and reliability for sensor networks.

4 Notes

Informal list of related works including: TOS (Mate / Bombilla, XMP), smart tags, push pin, stampede, MOAP, SensorWare, binary updates (efficient code distribution in wsn).