Wireless Sensor Networks: A Distributed Operating Systems approach

Paul Hunkin
University of Waikato
pwh4@cs.waikato.ac.nz

Tony McGregor
University of Waikato
tonym@cs.waikato.ac.nz

ABSTRACT
Wireless sensor networks and methods of developing user applications is an active research area with several unique challenges. In this paper we first present an overview of the current programming approaches. We then describe a system under development that adapts techniques from the distributed operating systems world to create a new method of programming wireless sensor networks.

Categories and Subject Descriptors
2.4 [Distributed Systems]: Network Operating Systems; J.8 [Computer Applications]: Wireless Sensor Networks

Keywords
Wireless Sensor Networks, Operating Systems, Distributed Systems

1. INTRODUCTION
Wireless sensor networks (or ‘WSNs’) are a collection of multiple, small, spatially distributed, embedded devices called ‘sensor nodes’. The nodes communicate with each other over a wireless network, usually in a mesh topology. The mesh typically contains a special ‘gateway’ or ‘sink’ node that uploads data to the Internet or other network.

Each node is equipped with various sensors that allow them to collect data about the surrounding environment. The core hardware and type of sensors can vary greatly depending on the intended purpose of the particular WSN, however the sensor node platform is typically a very low power embedded system, with small batteries as the power source.

A common wireless sensor node is the Scatterweb Modular Sensor Board. This has a MSP430 8mhz CPU, with 50k of flash and 5k of RAM. It has a low-power radio and is powered by three AAA batteries.

Much of the original work on WSNs was for military applications such as battlefield surveillance. However, additional uses have become common as the technology has become more accessible, with WSNs now widely used in various types of civilian environmental monitoring.

WSNs are an active research area, with much of the current work concentrating on the system software and middleware aspects. Several different approaches to sensor network system software have been developed, utilizing different programming models and levels of abstraction. The major types of approaches are discussed in the next section.

2. WSN PROGRAMMING
Because wireless sensor networks are a relatively new area and one that has significantly different requirements than most other areas of software development, no standard paradigm for programming WSNs has emerged. Many systems have been developed to run on wireless sensor nodes, most of which can be divided into three categories.

2.1 Single node operating systems
The first wireless sensor network applications were written to directly use the embedded hardware, with no operating system abstraction. While this approach makes efficient use of the limited hardware of the wireless sensor nodes, it was not particularly efficient in terms of programmer time and energy. Small operating systems soon became popular, some designed specifically for wireless sensor networks and some developed for generic embedded systems and ported to sensor node hardware. While these operating systems were very simple compared to more common OS’s such as Linux or Windows, they performed roughly the same role - to provide the programmer with a simpler abstraction than the underlying hardware, resulting in application software being easier to write and maintain.
The most widely used operating system for wireless sensor networks is TinyOS [11]. TinyOS is a component based operating system, written in NesC, a programming language based on C but which provides capabilities for the type programming required for TinyOS. TinyOS is based on the concept of connecting various types of components together, which provide abstractions for various operations such as reading from sensor devices or sending network packets. It is completely non-blocking and event based. Any operation that takes longer than 200 milliseconds must be implemented with callbacks. The operating system does not provide any method of threading or the conventional process abstraction, which can lead to added complexity when implementing longer running CPU intensive tasks. Several third-party extensions exist for adding lightweight thread support to the OS, however[2].

Support for conventional tasks/processes with blocking execution contexts is a long standing debate among WSN operating system researchers. If the OS supports these, it makes application programming significantly easier in many cases. However, this comes with a memory, processing and power overhead compared to a pure event based system, which is undesirable on hardware as constrained as wireless sensor nodes.

Despite this concern, several operating systems have been developed for WSNs that do provide various types of processes/tasks and blocking execution. The Contiki [3] operating system provides protothreads[4], a lightweight threading abstraction that saves memory by not storing the execution context of each thread. MANTIS[1] is another operating system for WSNs that provides more traditional threads, with pre-emptive multitasking and full context switching while still striving to maintain a small memory footprint of less than 500 bytes of RAM.

Applications written to run on all of these operating systems operate on a single sensor node - the programmer must be aware of this, with interactions with other devices explicitly specified in code.

2.2 Virtual Machines

Application virtual machines such as the Java Virtual Machine and the .NET Common Language Runtime have long been popular on both conventional consumer desktop and higher end server systems. They provide platform independence and program isolation. In recent years this technique has been used in sensor node programming to assist with the problem of reprogrammability - the ability to dynamically inject new program code into a sensor node in order to deploy updates to existing code or entire new applications. Virtual machines are a useful technique to provide this functionality as their bytecode is more easily moved, modified and relocated at runtime than traditional native machine instructions. The bytecode is also typically smaller in size than than native code, increasing the efficiency of moving application code over the network link.

Mate [10] is a stack-based virtual machines that interprets application-specific bytecode - the VM is designed for running a specific class of applications, which lets each bytecode instruction represent a relatively large amount of functionality. This technique necessarily limits the flexibility of the system, but lets the program be represented in a very small amount of virtual machine code, which reduces power and storage requirements.

VMStar [9] is a framework for allowing easy creation of application specific virtual machines similar to Mate.

Another use of virtual machines on WSNs is similar to the use on more conventional platforms - hardware abstraction, providing a common platform to write to across multiple hardware devices. The ‘t-kernel’ [6] is an WSN OS kernel implementing a virtual machine that provides this feature. It also provides additional features that are not provided by the underlying hardware such as memory protection and virtual memory.

2.3 Group-level abstraction

The concept of a group-level abstraction in wireless sensor network programming is to provide a method for dividing up the mesh of sensor nodes into logical groups, then performing operations on those groups rather than on the individual nodes. This technique hides the details of communications between the nodes. It is useful for implementing various types of ‘localized algorithms’, a term specifying operations where a sensor node is limited to interacting with only the nodes in its group.

Various criteria can be used to allocate node groups. The most common method is to use the physical location of the node - this works well to minimize network hops in sensor networks given their wireless mesh topology. Another method is to define the groups based on some other property of the node, such as the type of sensors or the energy levels remaining in the node. Groups can be static or dynamic, depending on configuration.

This approach can make programming sensor network applications simpler as less knowledge is needed about the underlying network and hardware. The cost is less flexibility in writing complex applications as the details of the lower-level system are not available.

2.4 Network-level abstraction

The network-level abstraction concept treats the entire sensor network as one abstract machine. There are two main approaches in this category - the database abstraction and macro-programming.

Treating the sensor node as a database is a useful metaphor as sensor nodes are often used to collect data. TinyDB[12] and Cougar[5] allow programmers to issue SQL-like queries over the sensor mesh, with intelligent caching, query dissemination and data acquisition techniques to minimize the power requirements of each query. The database abstraction is an intuitive interface to sensor nodes that is easy to use for simple data collection - a use case that covers a large subset of sensor node applications. However, these queries tend to be insufficient for implementing more complex systems.

Macroprogramming attempts to address this limitation by providing a more flexible method of creating sensor node applications that work at the network level. Regiment[13] is
a functional language similar to Haskell that is specifically
designed for macroprogramming WSNs. The functional lan-
guage choice makes it easier for the compiler to create the
node-specific code that eventually runs on the individual
sensor devices. Kairos\cite{7} is another similar system that
provides an extension to existing programming languages
with ‘var@node’ syntax to allow shared memory abstraction
across nodes. This lets programs that run on individual
nodes to have limited access to hardware on other nodes.

Another challenge in the network-level abstraction is the
problem of referring to the mesh resources. Spatial Program-
ming\cite{16} is a system that allows resources to be referred to
by their physical location - for example, ‘Hill1: microphone[0]’
for referring to the microphone sensor node on ‘Hill1’.

The various forms of network-level abstraction, in a similar
fashion to the group-level abstraction, simplify the task of
the application programmer. This again comes at a price of
flexibility - it is often difficult to implement complex applica-
tions that run on the mesh using the network level abstrac-
tion, due to limiting or cumbersome programming methods.

3. CONCEPT
The conclusion reached from a review of the existing WSN
application programming solutions reveals that each paradigm
has benefits and drawbacks.

The single-node programming model provides the maximum
flexibility in terms of writing applications that run inside the
sensor mesh. As the range of uses of WSNs is always increas-
ing, this flexibility is a major advantage. However the draw-
back of this approach is the increased complexity caused by
requiring the programmer to have knowledge of the sensor
mesh topology and manage the interactions between nodes.

Conversely, the group and network level abstractions are a
useful tool to simplify WSN applications, but sacrifices are
typically made in terms of programming flexibility - in many
cases systems are limited to simple data retrieval.

We seek to unify these approaches into a system that com-
bines the flexibility of the single-system programming model
with the high abstraction level and simplicity of the network
level abstraction. To do this, we have borrowed a concept
from the distributed systems world - that of the distributed
operating system.

3.1 Distributed Operating Systems
A ‘distributed operating system’ is an operating system that
is designed to make a collection of individual computers (or
‘nodes’) appear as one logical computer to the user. This
process is usually completely transparent - the user does not
need knowledge of what hardware they are actually using
or that a network is part of the system. When a user logs
into the system and executes programs, the decision of what
code to run on what piece of hardware is determined by the
operating system.

This approach has several advantages:

- Resource sharing - any hardware device on any indi-
  vidual compute node is (at least potentially) automat-
  ically available from any place in the system.
- Reliability - If a node fails or becomes slow, the system
  can recover by relocating tasks away from that node.
- Scalability - Scaling the system for increased workloads
  can be accomplished by simply adding more compute
  nodes at runtime.

Distributed operating systems date back to the 1970s, with
early operating system such as Arachne, CMDS and MI-
CROS first exploring the concept. It continues to be an
active area of research. Almost all Distributed OS projects
are implemented on relatively sophisticated hardware with
high energy cost requirements - unlike wireless sensor nodes.

3.2 The Hydra OS
Our system, which we have named ‘Hydra’ is a distributed
operating system targeted at wireless sensor networks. It
provides a familiar programming model similar to conven-
tional operating systems that abstracts away the unneces-
sary details of the underlying sensor mesh while still allowing
complex applications to be developed.

4. CHALLENGES
The nature of wireless sensor networks and the fact that
there is little previous work in this area means there is sev-
eral unique challenges involved in constructing this system.

- Hardware constraints - The hardware of a typical wire-
  less sensor node is very limited compared to the types
  of hardware that distributed operating systems are
typically implemented on, so all efforts to conserve re-
  sources must be taken. The system is being targeted
at the previous outlined Scatterweb MSB sensor node.
- Power constraints - sensor nodes have very limited
  power, normally low-capacity batteries. This means
that we must attempt to conserve power at all times.
As the most costly device by an order of magnitude
in terms of power is the radio, we must not send data
over the network unless absolutely necessary. This is
somewhat problematic for distributed operating sys-
tems, which inherently operate by having distributed
deVICES communicate with each other.
- Resource naming - the previously outlined problem of
  how best to specify remote resources such as sensors
or memory so as to remotely access them is an issue
here as well.

Of these, power optimization is arguably the most important
and most difficult - wireless sensor network software must be
constructed to conserve energy.
5. POWER MINIMIZATION

A key question is ‘How can we build a distributed operating system with a focus on power efficiency?’ Distributed operating systems operate by means of network communications between the hardware the current application is currently executing on and usually another remote node with hardware that is being utilized. In our case, the remote hardware will often be a sensor of some type. We wish to avoid network communications if possible. A useful method of accomplishing this is to dynamically migrate the application code to the hardware node that contains the sensor device in question. This would result in the sensor read operations being local, not remote - no network transaction is necessary, so the radio does not need to be used. The end result is being overall reduced power. Even if we cannot migrate the code on to the exact node, we may be able to move the code closer to the remote sensor node, decreasing the number of wireless mesh hops necessary. To accomplish this goal we propose two techniques.

5.1 Virtual machine

As outlined previously, virtual machines have become popular for their ease of reprogrammability and smaller bytecode making it simpler and less costly to move and update code from node to node. These characteristics are very useful to us for the task at hand, as the operating system will need to perform these types of operations relatively often and they must be as cheap as possible.

5.2 Application partitioning

The flexibility of virtual machine bytecode makes more complex operations possible. Applications can be divided into various logical objects and then operations can be performed on those instead of on the entire process. This introduces the possibility that a complex program that deals with a variety of different sensor sources can be automatically and transparently split into components that execute on different nodes. This finer granularity makes much more efficient power optimisation possible, and at the same time reduces the amount of program data that will be sent as the individual program objects will be smaller than the entire program.

6. IMPLEMENTATION DECISIONS

The criteria for selecting the virtual machine technology were, in order of importance:

1. The resulting bytecode is able to be partitioned as outlined above.
2. The implementation must be small enough in terms of code size and RAM requirements to run efficiently on the target hardware or the instruction set and associated standard library not be so complex that they cannot be implemented on the target hardware.
3. The instruction set is complex and flexible enough to implement the range of tasks the system might be used for.
4. If possible, a compiler and linker tool suite exists that can turn a high level language into the bytecode - this decreases the work necessary for the project. While the bytecode is all that the operating system will be concerned with, the existence of the high level language makes the application programming task much easier.
5. Ideally be a deployed, well known standard that can be adapted if necessary to the task at hand - this ensures additional tool support and increases the likelihood of the system being usable outside the research community it is developed in.

Many virtual machine systems were evaluated, from both inside and outside the wireless sensor network research community. The findings can be summarised as:

While the WSN community has produced several virtual machines such as Mate that have many beneficial features to WSN specific code, the majority of them are too narrowly focused on their specific research goal to fulfill requirements 1 and 3. These systems are application-specific virtual machines, which are built to support only a few individual user apps. They are not sufficiently general purpose to be usable for our purposes.

The few WSN-specific VMs that attempt to fulfill requirements 1 and 3 are still in relatively early stages of development. They use custom instruction sets and as a result do not have the tools support or deployment (4, 5) desired.

While many virtual machine implementations exist for conventional consumer desktop and server class hardware that fulfill all but 2, unfortunately the hardware limitations make them infeasible.

Taking these facts into account, it was decided that the simplest and most effective approach was to select an existing VM standard that exists on conventional systems. Then we would create a minimalistic implementation of the virtual machine to run on the MSB sensor node hardware. This strategy fulfills all the necessary criteria, with the downside that an initial investment of time to develop the virtual machine is necessary at the start of the project.

The standard eventually selected was Java.
6.1 The Java platform
While Java may initially seem an unorthodox choice for a wireless sensor network platform due to its reputation for high memory usage, it fulfills all of the points outlined above. The primary difficulty imposed by this choice is to implement the Java Virtual Machine (JVM) specification on the limited hardware available. Unfortunately few of the existing JVM implementations will operate in such a constrained environment.

An unofficial implementation of the Java VM called LeJOS[15] is the closest equivalent to our proposed system. LeJOS is targeted at Lego Mindstorm devices, which have sensor-node class hardware - 32k of RAM, with an 8-bit micro-controller. While it is unsuitable to use directly for Hydra due to its emphasis on the Lego hardware and environment, it does prove that the Java platform can be miniaturised to fit WSN constraints.

There are several techniques that can be used to decrease the footprint of the Java platform to make it suitable for the task at hand. These are:

- Discard most of the standard library - much of the footprint of the Java platform comes from the large standard library. Most of the functionality in the standard library is unnecessary for the proposed system - for example APIs for GUI programming, 3D acceleration, sound output and the like have little use on WSN hardware.

- Optimize the Java bytecode for size - there has been previous research [14] into optimizing .class files to make them more suitable for embedded systems.

- Internal VM optimisations - there are a range of techniques for optimizing the internals of the JVM, such as implementing standard library calls as native code so as to avoid the VM overhead.

One of the goals of the project will be to make this virtual machine as optimal as possible in terms of power and size. One of the research questions will be whether the benefit of the feature-rich platform is worth the overhead of the virtual machine.

7. JAVA FOR A DISTRIBUTED OS
Once the JVM is operational on the target hardware, the focus moves to the details of implementing the distributed operating system services on top of it.

Java helps accomplish the requirement of easy partition of user applications by nature of its object orientated design - programs are already divided into objects at the high level language and bytecode level. The system then can operate by moving the Java objects around the mesh based up on their power requirements and dependencies.

When the JVM detects that the application bytecode is attempting to invoke a method on an object, the JVM invokes the method locally or remotely depending on where the object is located in the mesh using some form of remote procedure call.

Power requirements for each individual object is primarily based on the number of network communications that object makes - both explicit network communications with the network functionality in the standard library or implicit communications caused by remote method invocations. This means that the power usage of each object can change dynamically based upon its position in the mesh relative to other objects.

8. RESEARCH TOPICS
The list of primary research challenges are:

- Creating a sufficiently complex VM on sensor node hardware - while not the primary focus of the project, the creation of a JVM that can execute the sophisticated bytecode necessary is a challenging task, and one that has not been accomplished previously to the degree necessary.

- Investigating the appropriate programming model for the system - event based vs threading, the method for addressing and locating sensor hardware in the mesh and similar problems.

- Determining the power requirements of bytecode - this will require testing different means of discovering the power requirements of both program objects and the program as a whole. While it would be useful to be able to perform this at compile time and save computation, it is simpler to do at runtime.

- Code placement based on power usage - once the power requirements of the bytecode is determined, the question then becomes one of placement. The optimal position for each object would have it requiring the minimal number of network transactions.

- Efficiently relocating bytecode across the mesh - once the position for an object is calculated, the next step is how to efficiently move it. The target is to seek to minimize the amount of network traffic needed to move the object.

In order to make a WSN programming system that is competitive with existing solutions, all of these problems must find an acceptable solution.
9. IMPLEMENTATION PROGRESS

The system is still in the early stages of development. The underlying kernel, network layer and sensor device drivers for the MSB node were adapted from an earlier single-node WSN operating system and are largely feature complete. [8]

The JVM is the current development focus. Current efforts are directed towards implementing the VM specification as a C library that can be compiled and executed for both the sensor node OS kernel and conventional Linux systems as a standard userspace application - the latter of which provides a more convenient development environment. The VM is complete to the point it can execute simple programs, though most functionality is not yet implemented.

The distributed operating system and WSN-specific functionality is not yet implemented. These features require the underlying VM to be more complete before they can be developed.

10. CONCLUSIONS

In this paper we have presented an overview of wireless sensor network programming models, and outlined the justification and initial design of Hydra, an operating system that is intended to adapt distributed systems programming techniques to provide a new programming model for WSNs.

11. REFERENCES


