Rewriting the Sensor Network Abstraction Stack

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. dr. ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op dinsdag 15 januari 2008 om 10:00 uur

door **Thomas Edward Victor PARKER**

electrotechnisch ingenieur

geboren te Londen, Engeland

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. H.J. Sips

Toegevoegd promotor: Dr. K.G. Langendoen

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. H.J. Sips	Technische Universiteit Delft, promotor
Dr. K.G. Langendoen	Technische Universiteit Delft, toegevoegd promotor
Prof. dr. H.L. Muller	University of Bristol
Prof. dr. ir. I.G.M.M. Niemegeers	Technische Universiteit Delft
Prof. dr. ir. M. van Steen	Vrije Universiteit Amsterdam
Dr. ir. P.J.M. Havinga	Universiteit Twente
Dr. ir. M.G. Maris	TNO

ISBN: 978-90-9022662-0

AUTHOR EMAIL: TOM@TEVP.NET

”If a man can write a better book, preach a better sermon, or make a better mousetrap than his neighbor, though he build his house in the woods, the world will make a beaten path to his door.”
- *Ralph Waldo Emerson (1803-82)*

Acknowledgements

This thesis could not have been created without help from many people:

- Koen Langendoen, my supervisor; for his confidence in my abilities that let me come to the Netherlands initially; for more help than I care to count with both the ideas herein and helping me through the insanities of Dutch bureaucracy; also for providing a voice of reason helping me come back down to Earth after I'd just proposed my latest wacky idea.
- Niels Reijers and Aline Baggio, my office mates, for listening to my ideas, and being interested while I explained why some very small item was vitally important.
- Gertjan Halkes, for all his help with debugging everything from sensor nodes to simulators (as well as correcting some of my larger coding errors).
- My defence committee, for helping this become a better thesis.
- My parents, for being understanding and helpful when I said I was leaving England, as well as throughout my PhD.
- And of course the cast of thousands who worked on the topics that helped me get to this work today, some of whom have listened and given me useful feedback when I explained what crazy things I was doing with their ideas.

Thanks everyone.

Tom Parker
Delft, December 2007

Contents

Chapter 1

Introduction

1.1 Origins

The origin of Wireless Sensor Networks (WSNs) can be traced back to the Smart Dust [58] project, which speculated about the possibility of computing systems at a micro scale (1-10 mm^3). Creating viable systems at this size brought up a lot of very difficult problems, notably how to build power storage and communication systems that small, but the core concept - lots (thousands to millions potentially) of small computers scattered into the environment - opened up a range of new applications, despite the technical difficulties.

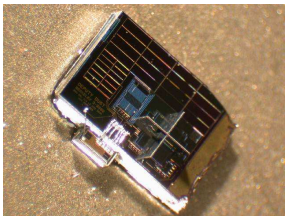
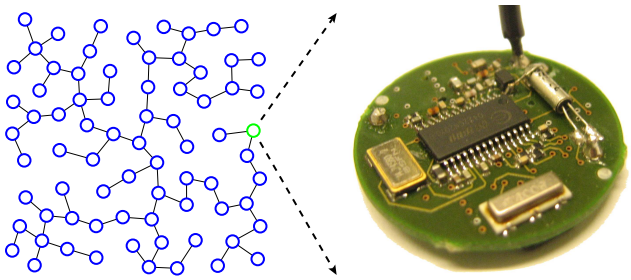


Figure 1.1: Smart Dust "Golem" prototype (6.6mm^3 volume)

The idea is that many cheap computers are used to cover a larger region than could realistically be covered by a smaller set of more expensive systems. Part of the intuition behind this design choice is the limited range of most sensors. For example, a temperature sensor gives you a single point value for a particular location. Buying a more expensive temperature sensor may provide better values, but it is still limited to a very coarse grained view of the surrounding environment. Therefore the primary way to improve the overall sensing quality is to use more sensors, and to place them in various different locations.

The data from multiple sensors then needs to be gathered in from scattered locations over an often fairly large physical area, and for many scenarios, placing lots of wires in the environment may well be impossible or interfere with the goal of gathering data

from the environment. Therefore, the notion of wireless communication, coupled with a simple computer is an appealing approach for data gathering from the sensors. Also, given that the quality of the data from the network is directly effected by how many of these wireless sensors can be bought given the budget of the system, making them as cheap as possible is a major design aim.



(a) Example network (b) TNOde ($\sim 900mm^3$ volume)

Figure 1.2: Wireless Sensor Networks

Thus, the field of Wireless Sensor Networks works with small computers (called "nodes") that are used to gather and interpret data about their environment. The current nodes worked with are much bigger than the Smart Dust ideas (compare for example the node in Figure 1.1 at $6.6mm^3$ to the node in Figure 1.2b at $\sim 900mm^3$) but the increased size allows for easier-to-use systems, while still being sufficiently faithful to the original concepts to allow for research into the problems that need to be dealt with by both sizes of node. Additionally, a greater number of applications become viable with larger nodes, especially as it now becomes possible to use off-the-shelf components (processors, batteries, radios) to build the nodes, rather than having to specially build every component, which can reduce costs significantly.

1.1.1 Applications

Working with many cheap wireless sensors scattered in an environment allows for a wide variety of new applications, and some examples of those that have been attempted so far are listed below:

- Precision agriculture
  - Precision agriculture for sensor networks has mainly focused on the gathering of micro-climate data for crop fields, in order to improve crop management decisions [18]. The LOFAR_Agro project ([39, 70, 134], Figure 1.3) has been attempting to reduce the required amount of pesticide used on a field by providing much more detailed information about the climate of a field, with the smallest resolved regions now being measured in meters v.s. kilometres with earlier approaches.

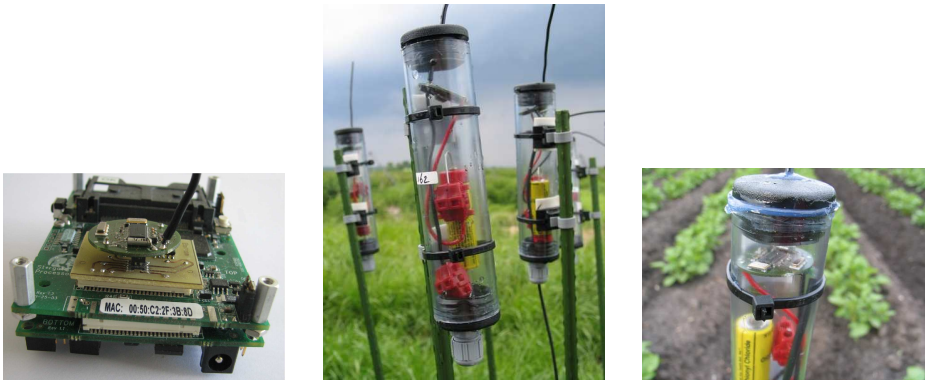


Figure 1.3: LOFAR_Agro project for precision agriculture

- Wildlife studies
  - The most notable wildlife study with sensor networks has been the Great Duck Island experiment [79, 131, 137], which gathered data on the Leach's Storm-petrel (a seabird that nests on Great Duck Island off the coast of Maine, USA). The Leach's Storm-petrel had previously been a difficult subject for zoologists to study, due to both adverse climatic conditions on the island, and its nocturnal lifestyle. With the aid of sensor networks, a detailed study of the species has been made possible.
- Architectural monitoring
  - Determining the stress levels in a building is a difficult task, and without that data, checking whether a building is structurally sound (because of earthquakes, or other sources of damage) is hard to test. Additionally, if a building becomes damaged over time e.g. dry rot, but it is not noticed by the inhabitants, then a building may become structurally unsound without warning. Several projects [20, 61, 117] have tested the idea of using sensor networks to record stress data for buildings over their entire lifetime, providing reliable and up-to-date data on the structural integrity of buildings.
- Military surveillance
  - A number of military applications have been proposed for sensor networks, including target tracking [6] (e.g. using magnetometers to detect tanks), sniper detection [123] (using sound measurements to track a bullet back to its source), and compound security [27] (detecting intruders around a specified perimeter).

- Monitoring cargo containers
  - WSNs can be used for various applications in the area of cargo container monitoring, both in the sense of determining where a cargo container is, and in the protection of a container from intruders. The most famous example of the latter is Sun Microsystems' "Project Blackbox" [109], which uses their SunSPOT nodes [1] to protect a cargo container full of more conventional computing hardware.
- Medical data gathering
  - Modern medicine relies heavily on data gathered from the patient to guide the choices of medical personnel. e.g. heart rate, electroencephalogram (EEG) readings, etc. Most of these readings are currently gathered by connecting a patient by wires to a (large) semi-static machine, which limits patient mobility, and the machines are often quite expensive. Sensor networks are being explored [88, 119] as a possible replacement, providing similar data in a cheaper and more portable form.
- Industrial monitoring
  - Monitoring industrial processes for changes in their performance can be used as an early indicator of equipment failure. As industrial locations are often unsuitable for additional cabling (due to space being at a premium), earlier work has used a "sneakernet" method of data transfer from sensors embedded in the sensors to analysis stations (i.e. physical transfer of data storage). This increases the time between failure being noticed by a sensor, and it being passed on to the operators of the system, and can result in an increased failure rate of equipment. Sensor networks have been shown to be a viable solution for reducing the delay time without requiring cable installation [64].

In short, sensor networks are a new tool used for many different applications, providing data over a longer period of time, larger locations, and with reduced costs v.s. other techniques.

1.2 A Different Approach

In order to create viable systems from many cheap sensors, WSNs are a hybrid endeavour, taking elements from many existing fields and combining them in new and interesting ways. The major predecessor fields are as follows:

- Distributed systems, with a focus on Ad-Hoc networking
- Embedded systems
- Wireless systems

These fields interact in complex ways when brought together. For example, most distributed systems research assumes both mostly reliable transport (which is removed by the wireless systems) and per-node resource limits similar to a desktop PC (which is not true in the embedded world, especially regarding power limits). This disruption of the common assumptions of the predecessor fields requires new approaches to be created for problems that would be otherwise considered solved.

In order so that it is possible to achieve usable results given the limits of the systems that the original goals specified, certain restrictions are made to the design space for WSNs, relative to the capabilities considered typical for other conventional related systems, and some new issues need to be considered.

1.2.1 Changing Roles

Instead of the arbitrary point-to-point networking of standard distributed systems, most WSN work focuses on a “source-to-sink” model of networking. This model of network communication incorporates knowledge about the typical design aims of a WSN, namely gathering data. The nodes that have data (“sources”), will not have enough power to transmit information very far, so external help is often required to transfer data to the end users. This is achieved by the notion of a “sink” node - few (often one) more expensive nodes with additional capabilities: bigger batteries, increased processing capability, and often a second network connection to a larger network (e.g. a wired connection to the Internet). Limiting the number of expensive nodes keeps with the goal of cheap networks, and as in general the major destination point for a packet from a source node is the sink node, this reduces the size of routing tables for each node as well as the amount of communication necessary to maintain said routing tables.

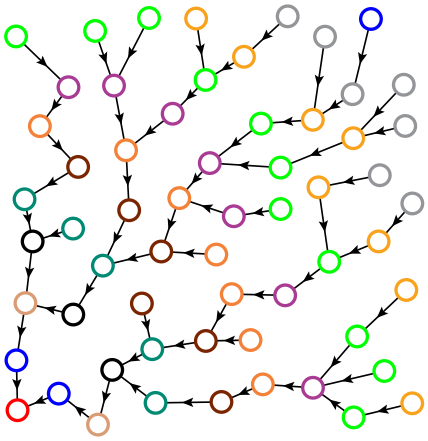


Figure 1.4: Source-to-sink networking

Regarding the data gathering role, applications for WSNs can be split into two major groups determined by the nature of their data gathering and many protocols are optimised towards one or the other role. They are *event-driven* scenarios, where data is only transmitted in the network when an "event" happens e.g. a person moves, or the temperature changes; and *periodic data measurement*, where data is generated at periodic intervals e.g. measuring temperature every 10 minutes. There are hybrid scenarios - event-driven situations often want a "heartbeat" message sent periodically (in order to maintain stable end-to-end communication links between times when event data is available); and periodic data measurement systems sometimes have a feedback mechanism to alter the rate of measurement in the event of significant changes in the data being measured - but most systems will primarily be in one or the other group.

1.2.2 Power issues

Power limitations are a major consideration for sensor networks, as the power resources (i.e. batteries) are generally scarce, and so there is the need to find a power efficient way to do things in order to have a network that works for a long period of time. The problem is that any attempt to actually do anything (like sending or receiving messages) costs power and so the most power efficient choice in every case is to switch off the node entirely. Sensor networks effectively need to do the minimum effort necessary in order to achieve the aims of a particular application, removing any redundant or extraneous effort wherever possible.

Figure 1.5: Limited power sources

One of the motivations behind power efficiency is the likely deployment of sensor networks in remote areas. If a sensor is deployed in a remote area, then going to that location and replacing the battery in every sensor (potentially hundreds to thousands) will be an expensive and time-consuming operation. To give an example of the problems that sensor networks face, we can note that the typical power available to a sensor node (600-3000 mAh at 3V with a pair of AA batteries) is comparable to a standard consumer mobile phone battery (typically ˜650-1500 mAh at 3.6V). Maximum lifetime for a mobile phone between recharges, even assuming that no phone calls are made, tend to not exceed approximately 10 days (with many not exceeding 3-4 days), which is comparable to the expected lifetimes for always-on sensor networks (estimates vary, but 4 days is typical [67]). In order to increase the inter-recharge lifetime to more viable levels (months to years), good power management techniques are required.

Indeed, the focus on working with only limited power, especially given that currently battery capacity per unit volume has not made any significant improvements in recent years [80], means that low power systems are and will continue to be a major research

aim in the sensor networks field. Although various efforts in the field of nanomaterials may well provide significant improvements in the coming years, which may cause new battery technologies to provide an order of magnitude or more improvement in power capacity, this is unlikely to increase the net available power. Given that for many current nodes the battery is the major contributor to both their size and weight, this improvement will probably be used mainly to reduce the size of nodes rather than to increase their power storage capabilities. This results in the conclusion that the total power available to any given individual node will probably not change significantly in the near future.

1.2.3 Locality considerations

The physical environment in which systems are deployed is also a more important issue for WSNs than for most other distributed systems. For most Internet-backed distributed systems, the precise location of a particular node is relatively unimportant. Coarse location data, with granularity down to the level of "which country" (even larger units are often acceptable) may be useful, but given the speed of most links in the modern Internet, knowledge of which nodes are the physical "neighbours" of each other is often not required. Some distributed systems protocols (e.g. Chord [129], Pastry [114]) have the notion of neighbour nodes, and may select these based upon latency or bandwidth values, but correspondence between these metric numbers and precise physical distance is often non-existent.

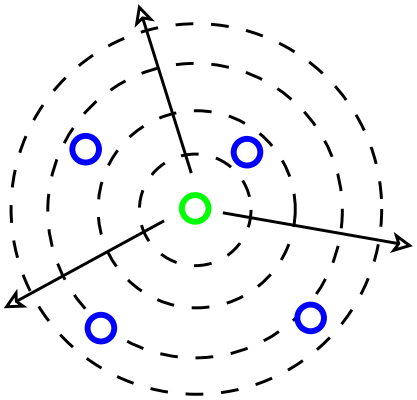


Figure 1.6: Radio-based neighbour nodes

For WSNs, physical locality is a much bigger issue. Most algorithms (in fact, all of the areas that we will look at in later chapters) will need some level of knowledge of which nodes can be considered "neighbours", and that will be based upon the ability of nodes to receive messages via their radios (see Figure 1.6 for an example of this), which is directly connected to physical locations and the environment around the physical locations. The requirement for knowledge of neighbours, combined with the list of the neighbours being determined by the local physical environment means that sensor networks are sensitive to where they are located. This applies both in a simple manner, in that radio reception rates will be effected by distances between nodes; and in a more complex manner, that the density of nodes in a deployed network is important, both if it is too small (not enough sensors in a given area to achieve application goals; reduced redundancy against bad connections/dead nodes) and if it is too high (many nodes competing for limited radio bandwidth). In fact, node density will directly effect both the capability of a network to achieve the application goals and the lifetime of a network [14, 53, 152].

Effective scalability of sensor networks is also an issue, partially due to the local-

ity issues. Firstly, in the sense of whether we can measure up to the original goals of "thousands of nodes" when most current deployments do not exceed 100 nodes. Even in simulations, most testing is done with less than 1000 nodes. Secondly, node density requirements will both limit the number of nodes of that can be deployed in a fixed area, and the maximum area that a fixed number of nodes can realistically cover without breaking down.

1.2.4 New Areas

In addition to the changes to old areas, a number of research areas that would not be considered for as part of traditional networking research are incorporated into sensor networks. The major additions are Localisation (Chapter 4) and Data Aggregation (Chapter 6), which are both responses to the issues specific to sensor networks - lack of (expensive) specialised hardware and the aforementioned power restrictions.

Localisation deals with the problem of determining the physical location of a node, and it is related to the locality issues mentioned above. The main purpose of a WSN is information gathering, and gathered data is only useful if you know what it applies to. For example, the data "the temperature has gone up by 10 degrees" is not very useful, but the information "the temperature has gone up by 10 degrees in room 3C" is a lot more interesting. Location information gives us a context, which allows us to actually use our gathered data. In other systems, GPS would be an option, but given the relative costs of GPS units (which are comparable on their own to the costs for a sensor node), the power requirements, and the difficulty of using GPS indoors [76], sensor networks need new solutions to the problem of Localisation.

Data Aggregation seeks to take multiple packets from nodes and combine them into a smaller set of packets before transmitting them further into the network. The intuition behind this is that if less data needs to be transmitted, then it will consume less power. Of course, this requires discarding some of the data, so aggregation techniques need some information about the nature of the data and what the application requirements are for the use of the data, in order to avoid discarding the important data.

1.3 Problems

With multiple complex goals in mind, and needing to work within the restrictions of the application scenarios, WSNs naturally have a wide variety of problems to deal with, a number of which appear to be related and spread across the entire field. This would imply that there is an underlying problem that needs to be looked at, and we set out to try and discover what that is.

1.3.1 Treading on the fingers of giants

Having a substantial body of existing work to start from has its advantages and disadvantages - on the plus side, WSNs get a number of existing protocols to work from; on the down side, existing ad-hoc networking protocols were not designed taking into account

the design restrictions of WSNs. To combat the problem of protocols not being power aware (i.e. the primary difference between WSNs and other distributed systems), most work in the field of WSNs has focused on finding more power efficient solutions to the existing problems. This is a good approach, but many of the standard problems being focused on are not the actual problems.

For example, for routing, the standard problem is "what is the shortest-hop path from source to destination?". For wired networks, or for wireless networks without power limits, this is a perfectly good question. However, it is one step away from the actual problem, which is "how do I get my packet to the destination node?". If you have an answer to the former question, then you have a solution to the latter question, and so people get confused between the two. For sensor networks, with their limits, this is an expensive mistake. The first question can be answered in a WSN context, that answer can be used to achieve the application aims (get data to a sink), and more and more optimal solutions to this problem can be found that will improve the energy efficiency of the system. But, stepping away from the first question, and instead dealing with the full problem directly gives you a wider range of possibilities, and allows much more energy efficient solutions i.e. finding the most energy-efficient path, as opposed to the one with the smallest number of hops, as the two are often different given unreliable links between nodes.

One of the reasons why it is easy to fall into the trap of answering the wrong problem is how the problems are viewed. Going back to the routing example, the idea of a shortest-hop path is a simple way to view routing - this idea of a particular path for packets works well in people's minds. Moving from that simplistic model with "perfect" links to a more detailed model with unreliable connections is harder to visualise, and so it is difficult for programmers to envision the complete set of actions trivially. To some extent, this is another example of the original statement - being one step away from the problem - as the problem here is not how can a set of nodes implement an algorithm, but how can programmers sufficiently incorporate the algorithm into their world view in order so that they can understand how to write the code in the first place.

In order to find how to solve the right problems, we need to look in more detail at why we pick the wrong problems, and then how we can tell the difference between right and wrong problems. Some effort has already been done in the field of challenging standard computer science metaphors, both in general [57] and in distributed systems specifically [128], but more work is needed dealing with the relations of the issues to the scenarios commonly encountered in sensor networks.

1.3.2 Abstract software

In the software realm, everything is an abstract construct, and often a construct based upon an entire series of other lower layer constructs. Even the things we regard as a physical object (CPUs for example) are themselves abstract concepts. A CPU is an abstract concept that we use to describe certain high-level effects generated by a series of smaller physical objects (processing units, logic gates, transistors, electrons) grouped together in such a way that they behave in ways that conform to certain models that are useful to us.

A model is useful in the sense that it is an abstraction away from the complexities of the full system e.g. what every single transistor in a CPU is doing v.s. what the CPU is doing. The full system simply has too many details, and in order to facilitate reasoning about the system, we think about things in a particular way (a "mental model") in order to help build a simplified model of the full system i.e. an abstracted view of the full system.

Reddy [107] and Lakoff [68] showed that the choice of mental models is intrinsically tied to how people think about the world, and looked at this for the general case. Their work with conceptual metaphors showed that people mentally model abstract concepts based upon experiences of physical events, describing them as "experiential metaphors", and we can apply this line of reasoning to computing-related abstractions as well.

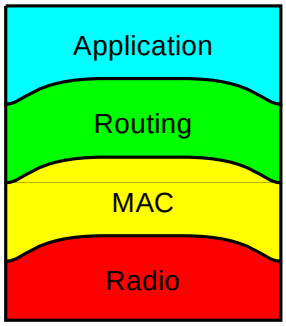


Figure 1.7: Part of the CPU abstractions

Many of the physical events associated with mental models of computing concepts involve one or more people doing a task that we would like the computer to do, e.g. the common abstractions of a "stack" of objects being like a stack of cards; the notion of a "queue" of tasks being like a queue of people; the entire field of "agent-based" computing. For simple examples like queues and stacks, the connection between the abstract concept and the physical example is obvious and clear. Assumptions and extrapolations based on knowledge of characteristics of the physical example have direct analogs in the abstract concept e.g. we can add more people to a queue, remove people from the front of a queue, and the same problems occur with multiple queues (some queues may empty faster than others for example) in both the abstraction and the physical example.

When we start to move to more complex examples e.g. sending a packet from one node to another as being like sending a letter in the post, a greater level of detail is lost between the abstraction and the physical example being used for our mental model, and this starts to cause problems when attempting to reason using this model. In the mail example, a piece of mail is only held by one person at a time, and it is impossible to receive multiple copies of the same piece of post (a sender can send multiple copies of an identical message, but they are separate pieces of post), whereas it is possible for multiple nodes in a computer network to hold copies of a packet, and in some situations a node will receive multiple copies of the same packet. Similarly, all postal mail is implicitly unicast, with no concept of "broadcast" mail, but we can broadcast data packets to multiple receivers.

One way to remove the problems with inconsistencies in our mental models is to build better models. Instead of treating data packets like postal mail, we can think of moving data packets around a network as being like people exchanging a message by talking to each other. Multiple people can know a message, thus resolving the duplication inconsistency, but now we have to assume that the messages are passed from person to person correctly. In the computing system, this is achieved by the use of CRC calculations, but this is infeasible for human-to-human communication. We created a new mental model,

but once again our model was flawed, as there were inconsistencies between the model that we use to reason about the abstraction and the complex system we are attempting to model, and this will lead to flawed thinking regarding any use of the abstraction.

Furthermore, the initial creation of a new abstraction will be based on a mental model of a system, and so flawed mental models will result in flawed abstractions. This occurs when a programmer is working with a complicated system, and believes that a higher level of abstraction will help remove unnecessary details from his/her view of the system. The notion of which details of the system are "unnecessary" and which are most important is part of the mental model of the programmer when he/she is building the new abstraction, and so flaws in that model will be reflected in the abstraction.

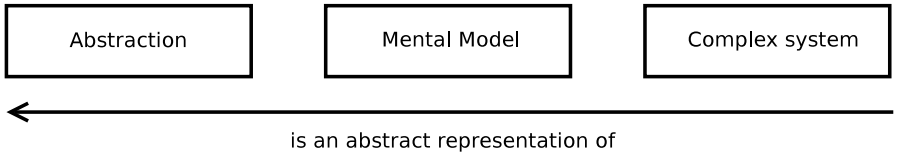


Figure 1.8: Levels of Abstraction

Deriving techniques for building better mental models is a difficult task, and more a matter of psychology and pedagogy than computer science, so our focus in this thesis is on working with examining and rebuilding abstractions.

One problem that may limit what we can do to improve our abstractions is that it has been observed in general that "All non-trivial abstractions, to some degree, are leaky" [126] i.e. the supposed gain from abstraction - not having to know about the underlying system supporting the abstraction - is significantly imperfect, and knowledge of the underlying system is often required to be able to understand how to fully use the abstracted concept. Also, every abstraction that is used has a cost (often implicit rather than explicit), in the sense that it is generally always possible to build more efficient things by working without that abstraction layer.

A possible conclusion from all the problems with abstractions is that the most efficient results can be gained by designing every physical object from the quarks up, and that software should not exist in favour of custom designed special-purpose hardware, which has again been optimised to be perfect for the particular task in hand. Unfortunately we cannot take this approach, as it is impractical for a number of reasons; the most important being that for all non-trivial aims this would take vast amounts of time and effort, in part due to the effort of working with complete models rather than abstractions. We therefore need to discover a level of abstraction that is suitable for the systems that we want to develop.

1.3.3 Improved choosing

The conflict of tending to think about things based on abstract metaphorical mental models derived from arbitrary experiential data v.s. the reality of the systems that we are dealing with being considerably different to these models (despite our attempts to build an abstraction that resembles our thinking) results in variable (often high) costs.

Therefore, we need to select a good middle ground, that balances between more abstraction, with a base set of primitives to work with that are closer to how we think about things (and therefore easier to work with), and the cost of that abstraction. All abstractions take items from the lower layers and build new objects that can be worked with, but the decisions about how to do this grouping vary considerably. Not all abstractions are equal either in terms of cost or in ease of use. The cost of an abstraction is mainly down to two factors:

- Structural resemblance to lower layers - does the abstraction work with or against the lower layer structures? Working against is more expensive, but may provide a structure that is easier to work with.

- Size of groupings - does the abstraction group many lower level objects, or only small subsets? If there is a lack of structural resemblance, then the groupings are often larger, and may stop users of the abstraction who do not need to use all of the items in a group together from doing tasks efficiently. Smaller groups however often provide a reduced gain in ease of use as the abstraction is similarly easy to use as the underlying layer.

A good example can be given using programming languages. If a solution to a task is created using Lisp (while working with conventional processors), then it will not be the most efficient design possible due to the lack of structural resemblance between Lisp and the underlying layers (C, assembler, processor instructions). A design in another similarly high-level language (e.g. Python, Perl, Ruby) may well be able to be more efficient. These other languages also differ considerably from the lower layers, but given their closer structural resemblance, they have a lower abstraction cost.

However, the best choice is very much application-dependant. If the problem matches well with the semantics of Lisp, then although a more efficient solution may be possible, the Lisp solution may well be better than the solutions that could be achieved in the other languages with a similar level of programmer effort. Conversely, for many applications (or at least parts of applications), designing in a lower layer language (e.g. C) may well be a better balance between efficiency and programmer effort.

The critical piece of knowledge is to be aware of the abstraction cost. With software, this is often not very obvious, and easily forgotten. In other fields, the abstraction is often obvious - for example, the use of foundations in architecture allows architects some degree of abstraction away from the problem of whether they're building for clay and chalk soils that need minimal foundations; or peat and bog soils that require deeper foundations, allowing them to focus on the issues of how to build the house on top of the foundation. With software, abstractions have often been built by other programmers, and so the costs appear as reduced speed or increased memory usage of the system, as opposed to being costs of the abstraction layer, and separating the two is often non-trivial.

1.3.4 Sensor Networks

Much of what has been stated here regarding the notion of abstraction is not a new problem for computer science, as many years of research into programming languages and the extensive debates regarding the trade-offs between higher and lower-level languages will attest. Abstraction has not however been suitably dealt with in sensor networks, as later chapters of this thesis will show that the trade-offs intrinsic to the selection of a suitable abstraction for a particular subsystem have not been properly analysed at all levels of the sensor network stack.

For systems where plenty of power is available, e.g. the desktop PCs, servers and human-portable devices focused on by the rest of distributed systems research, a trade-off that emphasises ease-of-use and creates a high-level abstraction is acceptable because it provides substantial reductions in programmer time and effort. These are the predecessor fields to sensor networks and they are where we have inherited our abstractions from, without any significant amount of thought whether they are at all suitable for sensor networks.

Sensor networks are not suited to choices that emphasise ease-of-use over efficiency. The major factor distinguishing sensor networks from other fields is the reduced resource budgets (energy, processing, memory, etc), and so the logical conclusion is that the trade-off between ease-of-use and efficiency needs to be revisited. This trade-off has been correctly considered regarding the choice of programming languages - no one, to the best of our knowledge, is trying to write programs in Python or Perl for sensor nodes, and the majority of code is written in variants of C - as the ease-of-use/efficiency trade-off in programming languages is familiar to most computer scientists, but there is far more that can be gained by looking beyond simply choosing appropriate programming languages.

1.4 Contributions of this thesis

In order to achieve the levels of efficiency necessary to get usable lifetimes with limited energy resources, and to get good results out of WSNs, some of the less obvious abstractions need to be challenged - those that are not even considered as abstractions - and examine whether better abstractions can be built, or if the trade-offs are appropriate to try dealing with the system at a higher or lower level. Re-examining the “abstraction stack” will allow models of thinking about sensor network problems that are more efficient than anything we could achieve with the current mental models.

In this thesis, we will examine various major groups of sensor network protocols, show how earlier work has (mis)used abstraction, and demonstrate how an improved model can be derived by re-thinking the level and nature of abstraction in the protocols. In each case, we provide an example of an improved protocol design that uses the proposed improved model, and show the gains over traditional models for the protocol type. Many of the different protocol types for sensor networks have an implicit assumption of a “building block” - something that is at the core of most protocols of that type, without

ever being thought about in much depth. Re-examining these building blocks is the main focus of the work here.

The various different layers that we will examine interact with each other as shown in Figure 1.9 to provide a complete software stack for WSN applications. Other possibilities exist for interaction within sensor network stacks and different choices of layers (e.g. cross-layer designs), but the majority of sensor network protocol design interact as shown here. We evaluate our new views about each layer using a combination of simulation models and/or experimental validation of the protocols that use the improved abstractions (and comparisons to protocols that use the "standard" abstractions where they exist).

Notably, although the predecessor fields to WSNs have sufficient resources to use the inefficient models, the more efficient models proposed here will in some cases also be suitable for use in fields outside of sensor networks, as even with abundant resources improved efficiency is useful.

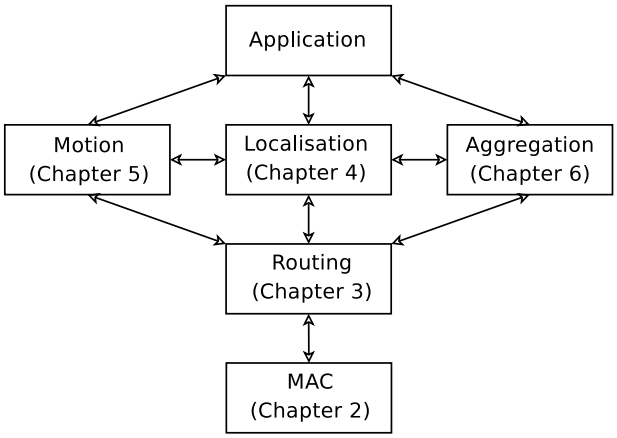


Figure 1.9: A typical WSN software stack

In each chapter of this thesis we take apart the assumptions of a different layer, with the chapters laid out as follows:

- MAC protocols (Chapter 2) - we take apart the notion of a MAC protocol as a low-layer, radio-dependent system, and build an improved modular framework for constructing MAC protocols
- Routing (Chapter 3) - we deconstruct the idea of "unicast links" between nodes, build a new set of sending primitives, and use them to build an energy-efficient routing protocol
- Localisation (Chapter 4) - we challenge the concept of distance estimation between nodes, define "probability maps" for distance estimates, and build a localisation protocol that can handle inaccurate ranging data using probability maps.

- Motion (Chapter 5) - we further re-examine the abstractions developed in Chapter 4 (probability maps and bounding boxes), look at differential probability maps, and build new protocols that can do motion detection both with (Portmanteau) and without anchor nodes (Adumbrate), but without requiring motion-detection hardware.
- Aggregation (Chapter 6) - we challenge both the use of standard statistical functions for aggregation, and the notion that aggregation can always combine all data into a single packet. We then build a phase space representation for arbitrary application-specific data, and build a new aggregation protocol that uses the phase space representation to significantly reduce the errors v.s. traditional aggregation protocols.

Chapter 2

MAC Protocols

In this chapter: We take apart the notion of a MAC protocol as a low-layer, radio-dependent system, and build an improved modular framework for constructing MAC protocols

Most non-trivial networking systems, at the physical level, work with the idea of using a medium that physically connects nodes of the system, and that is used to exchange messages between them. WSNs are no exception, and in this case we are mainly working with radios, with the environment around the nodes being the medium. For radios, that medium is shared, with many nodes potentially needing access to the same part of it at the same time, so we need protocols to be able to decide when a particular node can send messages and when it cannot. These are termed Medium Access Control (MAC) protocols, and in this chapter we deal with the choices made in designing these protocols for the WSN field.

Physically speaking, for radio-based systems, there is nothing stopping two nodes from trying to use the medium at the same time, but depending on the exact locations and orientations of the nodes, as well as the surrounding environment, the results of doing so will vary significantly. Completion of the transmission of a message is a relatively easy task, and the only thing that will generally stop this from being done is other usage of the radio by software on the node, which is information that software on the node is capable of having complete knowledge of.

Correct reception of a packet by the intended destination node(s) is far less certain. Knowledge of the state of the radios of other nodes, or the effects of those radios to the environment surrounding a node, is necessarily imperfect, partly due to the quantity of unknown and changing values (such as objects around the nodes that may absorb or reflect radio waves), and partly due to the problem that generally the only way to share information about the radio of an individual node is via the radios. Additionally, future packet rates are often unknown by the MAC protocols, as this information is a result of the interaction between application and routing layers (and possibly localisation, aggregation, and other protocols as well). Even in the degenerate case with only an application layer above the MAC, applications do not necessarily know in advance exactly when they

will need to send data (especially for event-based systems, see Section 1.2.1 on page 5), and no current systems (either for simulation or real nodes) provide a mechanism for the sharing of this information.

MAC protocols therefore need to attempt a trade-off between sharing additional information between themselves and creating some level of synchronisation on who sends when, v.s. the cost (energy, time) of this information sharing and the benefits that can be gained given the above issues regarding imperfect information. Although transmission is a simple matter, transmitting at the right time to make reception possible is a more difficult matter given the number of influences external to the sending node that would need to be considered, and that information about these will always be partial. Additionally, due to the lack of perfect information, no individual packet can be assumed to have successfully been received. However, the use of retries, redundancy, acknowledgement packets and of course careful use of the information that is known, can improve the odds significantly.

2.1 MAC concepts

Before we look at the existing work in the field of MAC protocols, we need to look in more detail at how the problems that we have already stated have been considered in WSN MAC protocol design. As we stated in Chapter 1, our thinking about software is as abstractions from the real problems, in order so that we can try and piece together a mental model that approximates the problem without overloading us with too much information.

Some of the common abstractions used within existing MAC protocol design are:

- Collisions - two nodes both sending at the same time is abstracted to the concept of two physical objects colliding with each other (which is related to the "packet as physical object" abstraction which we will look at in more detail in Chapter 3). The two sending nodes do not strictly speaking "collide", but the effect of two signals both reaching the same point in space has an end result that is different from either initial signal.

  We then also have two special cases of collisions which are often dealt with as separate problems

  - Interference is when a (weaker) second signal is able to reach the same point in space as another signal, but the second signal would be insufficiently strong to be received even if the first signal was not present. The separation here between the notions of collision and interference underlines the faulty abstraction mapping between the reality of multiple signals being transmitted v.s. our attempts to create mental models.
  - The Hidden-terminal problem [136] (as shown in Figure 2.1) occurs when two nodes (A and B) want to send to a third node (C), and neither A nor B can

hear each other's messages. As neither A nor B can hear each other's messages, they are each unaware that the other is sending, and so they will collide. This is opposed to situations where two potential transmitters can hear each other's signals, and so the second (chronologically speaking) sender will hear the message of the first and not transmit its messages. The hidden-terminal problems usually refers to situations where both messages would have been sufficiently "strong" to be received by the intended destination node (C), but the "collision" stops this from happening.

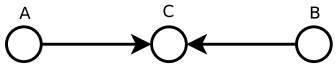


Figure 2.1: Hidden-terminal problem

- Idle listening is any time when a node is listening to the radio, but no messages are currently being received. This is to a certain extent, a WSN-specific MAC problem, as idle listening does not interfere with other radio traffic, but it does consume power, which is a much bigger problem for WSNs (with their power limits) than for other systems.
  This can be inferred from the power consumption for a typical WSN radio [21], which for reception at 868MHz typicallly consumes 9.6 mA v.s. 0.2 μA when asleep. In other words, the radio consumes *48000* times more power when listening than when asleep, and in the low-traffic scenarios typical to WSNs idle listening actually consumes much more power than transmitting packets, so WSN MAC protocols tend to optimise towards reducing idle listening.

Of these abstractions, the notion of collisions is most interesting, because of the reasoning behind why it has been constructed, and why it was regarded as necessary (if only as a default assumption given what we discussed in Chapter 1 regarding the choices that guide our abstract thinking). In effect, collisions and its related abstractions (interference, hidden-terminal) are an attempt to move away from the purely analog nature of radio, and towards a digital expression of the concepts.

Although the analog radio signal is converted to/from a digital signal inside the radio transceiver hardware, the signal itself remains fundamentally analog, and the result of multiple signals colliding is also an analog result. Specifically, a mixed signal from multiple sources is also capable of demonstrating the patterns that the digital signal conversion is looking for, such as in the BitMAC protocol [111], which uses the notion of multiple bit-synchronised On-Off Keying [144] signals with an emphasis on an analysis of the similarity of their likely combined form given the analog nature of radio signals to the original individual signals. Alternately, Black Bursts [124] uses secondary characterisitics of the messages (the length of messages) rather than the data within them to also be able to work with collisions, by noting that multiple colliding messages will create a garbled sequence of data of length equal to the time between the beginning of

the first message and the end of the second. Both of these directions for protocol design require stepping back from the standard abstractions and re-examining the nature of the problems that we face.

2.2 Implementation Difficulties

Instead of attempting to find a new way to rebuild the building block abstractions for MAC protocols (e.g. finding a new way to view the hidden-terminal problem by analysis of the underlying radio signal issues), we decided to re-examine the components used in the design of more conventional MAC protocols, as our earlier experiences have shown that getting even a fairly conventional MAC protocol to work correctly on node hardware was much harder than we expected. Discussions with other MAC designers indicated that this was not a problem with our choice of MAC protocol, but a result of the creation process for a MAC protocol at this time requiring direct interaction with low-level components of the system.

Interacting with hardware components such as radios and dealing with their interactions with millisecond resolution clocks on typical WSN node hardware is difficult, and this level of difficulty is something that a good abstraction process should have reduced significantly. This indicates that there are concepts within the MAC protocol design process that are not properly abstracted i.e. we need to build new abstractions to encompass areas of the design space that should have been thought about in a more abstract way; or possibly there are existing "buried" abstractions that we have not yet been able to identify correctly, and can probably be used more efficiently once we are fully aware of their presence within our existing processes.

As there appeared to be room to reduce the effort required for building MAC protocols by re-examining the level of abstraction used, we decided that further exploration into how MAC protocols are built was required.

2.3 Types of WSN MAC protocols

In order to properly rebuild the implementation process for a MAC protocol, we needed to look what the range of possibilities was. Given all of the problems for MAC protocols, both the more abstract forms, and the core issues underlying them, a lot of thought has gone into their designs, and many different approaches have been considered.

Current WSN MAC protocols are usually grouped in two different broad categories [71]: Time-Division Multiple Access (TDMA) protocols (TRAMA [104], PEDAMACS [23], LMAC [50]) and Carrier-Sense Multiple Access (CSMA) protocols (B-MAC [99], Wise-MAC [33], Sift [55], as well as hybrids usually described as CSMA such as S-MAC [148] and T-MAC [24]).

2.3.1 TDMA

TDMA protocols work by dividing the available time into "slots" and "frames". A slot is a (small) period of time, in which only a subset of all the nodes (usually one in each local "neighbourhood") are allowed to send messages, and a frame is a longer period of time made up of a series of slots. Most TDMA protocols attempt to create an environment where only one node in any particular area can send at any one time, which should eliminate the hidden terminal problem.

LMAC [50], for example, does this by allocating slot numbers where a node is allowed to send messages (defined as the position a slot occupies within a frame) such that they are not re-used within a two-hop neighbourhood, so neither a node's neighbours nor the neighbours of its neighbours will have the same slot number (in order to get around the hidden terminal problem [136]), and therefore any messages sent by a particular node should never collide with messages sent by other nodes.

Some TDMA protocols (e.g. Crankshaft [42]) allocate when nodes are allowed to send to a particular node as opposed to which node is currently allowed to send, but the core principle remains the same. The difference is that rather than allocating slots for when nodes are senders, and allowing all other nodes to be receivers; instead slots are allocated for when nodes are receivers, and all other nodes can be senders to the receiver node.

2.3.2 CSMA

CSMA protocols allow for sending at arbitrary points in time, but before a node may send it needs to perform a "carrier sense" operation by listening to the medium for a (usually) short period of time to check whether any other node is currently using the medium before starting to send. The IEEE 802.11 MAC [52] is the most prominent example of this technique, but it was not designed with low power usage in mind, and so is not suitable for most sensor network nodes.

Another popular technique in this area is Low-Power Listening [32], which attempts to reduce idle listening by coupling a long preamble with frequent short carrier sense periods that are able to detect the preamble bytes. Preamble bytes are normally used in MAC protocols at the beginning of a message to "train" the receiver (by synchronising the carrier waves, see [105] for more details) to more accurately receive a packet, and are generally only a few bytes long. In LPL, the preamble is much longer ($>$1000 bytes in some cases). The idea is that a preamble of n bytes long can always be detected by the short carrier sense periods, provided said periods are no more that n bytes apart.

LPL is more expensive for the sender node, but less expensive for receiver nodes as they do not need to continually sample the radio medium, and as in general there are more receivers than senders, it is designed to reduce total energy consumption in the network by reducing idle listening. This was later expanded into the WiseMAC [33] protocol, which uses past knowledge of the receiver state to decide when it is a good time to start a preamble sequence, allowing for much shorter preambles than LPL.

There are also hybrid protocols which use elements of both CSMA and TDMA. For example, S-MAC [148] and T-MAC [24] make nodes wake and sleep for periods of time, with the exact intervals determined by one or more shared schedules, similarly to the shared time notions from TDMA protocols. During awake periods, carrier sense methods (as for CSMA protocols) are then used to determine when a node can send.

2.3.3 Differences between the types

TDMA and CSMA approaches are usually regarded as being very different, and even within each approach we see many different protocols that all do things in significantly different ways. Despite all the apparent differences, all of these protocols have one thing in common - they are designed to manage the available time in the radio medium in a way that attempts to optimise for particular useful metrics while sending/receiving messages. The notion of which metric (latency, energy usage, etc) is most useful is application-dependant, and often several metrics will be optimised for at the same time, with some being regarded as more critical than others.

They all do this by managing when a particular node can send messages - TDMA protocols do this by separating the available time into slots and allowing nodes only to send in their slot; CSMA protocols do this by making nodes perform carrier sense before sending (and in the case of protocols like S-MAC, also by waiting until the beginning of the next “frame”). In total, a MAC protocol must do two things: given an application wishing to send a packet, determine what time this node will be able to send and send the packet at that point; and transmit appropriate control packets so that the application layer will be able to send packets in the future.

2.4 Problems

We are now better able to explain why building MAC protocols was harder than would have previously been expected. Current MAC protocol design for WSNs covers a wide variety of different tasks, in addition to the core item of managing when to send messages. A MAC protocol is regarded as being responsible not only for deciding when to send packets, but also what to send. For example, generating the standard Unicast sequence of RTS/CTS/DATA/ACK messages is usually the responsibility of the MAC protocol after the application has provided a data packet to be sent. The MAC must maintain an internal state machine monitoring which one of these packets it last sent or received, enabling it to determine what packet should be sent/received next. This state machine is not particularly complicated, but when intertwined with the timing mechanisms required by the MAC, the complexity of the combined code is often much greater than simple addition of the complexity of the two separate code paths might suggest. Additionally, as much of this is common to all MAC protocols, there is duplication of functionality, which leads to MAC protocols code size being larger than necessary (which is related to the code complexity increase [15]) and therefore increasing the probability of higher numbers of bugs.

Another problem is that the decision about whether a MAC's implementation of Unicast uses RTS/CTS messages (which are seen by some designers as overhead, and by others as required for reliability) tends to be a somewhat haphazard affair. Often, whether their additional reliability is required should be a decision made above the MAC protocol level - better choices include at application or routing level - and so some MAC protocols that implement RTS/CTS allow this functionality to be switched off and on at run time. The possibility of this option existing is an example of a feature that may or may not be in a given MAC protocol depending on the whims of its designer. Additionally, extensions to these basic functionalities must be separately implemented in each MAC protocol.

Given that we have a set of functionality that should be common to all MAC protocols, but certain implementations do or do not have particular features implemented, we lose out on a major advantage of common functionality: the idea that we can ideally use any given MAC protocol as a drop-in replacement for any other. If we could in fact easily swap MAC protocols, then application designers would be much more free to choose the protocol that is most optimised for their needs, as opposed to the default MAC built into the system. Additionally, because the duplication of effort results in both increased bug count due to multiple implementations of the same ideas (e.g. Unicast), and a system that is hard to extend, we conclude that our initial idea that MAC protocols needed redesigning was correct; in that the current standard design brief for MAC protocols has a number of significant problems, and therefore it should be rethought.

2.5 A new MAC stack

Given these problems, we wish to redesign the process for creating a MAC protocol such that the common functionality that does not necessarily need to be in a MAC protocol itself can be separated out. The first step to achieving this is to determine what is common functionality, and what are MAC-specific requirements.

We looked at separating the existing large MAC protocols into 3 parts: below the MAC, above the MAC and a "λMAC layer", which would compromise the core "true" role that should be the part of the code that reflects the choices of the MAC designer. This set of layers we refer to collectively as the MAC stack, and together they should do everything a traditional monolithic MAC layer would do on its own.

2.5.1 Underlying Modules

Several modules are required "below" the λMAC layer. Working from the conclusions of Section 3.2, we know that MAC protocols need to send/receive packets, and to decide when to send/receive. The first can be achieved with a "dumb" packet layer (no queueing, minimal latency, switches radio on/off only when told to); the second requires medium activity detection (as part of the "dumb" packet layer) and/or a time synchronisation layer. Time synchronisation can also then be used to generate "frames" (periodic timers, as used by all TDMA protocols and S/T-MAC), but it needs to be designed such that it will not interfere with protocols that do not require time synchronisation (e.g. B-MAC [99]).

2.5.2 Transmission Layer

The biggest question regarding how much we can pull out of a standard MAC layer was deciding what a λMAC layer actually really needs to do. Or in other words, knowing what a complete MAC stack needs to do, what makes one MAC protocol different from another? Our conclusion was time management. One of the standard opinions about the role of WSN MACs is power management, and time management can be considered an extension of this - one of the time management roles is deciding when to switch the radio on/off, but another is deciding when to start sending a packet sequence. However, once a node has started a packet sequence (e.g. all of Unicast after the RTS message), the code becomes remarkably generic and MAC-portable, yet is currently still embedded within the MAC. What if we could extract that - let the MAC decide when to initiate packet sequences, but then hand off to a generic module to perform the actual sequence itself? This new *transmission* layer module could then be reused in other MAC protocols.

2.5.3 Time Management

Now that basic packet sending/receiving, time synchronisation, and the sending of particular packet sequences have all been separated out, the λMAC layer only needs to contain time management: that is, the maintenance of the knowledge about what time is a good time to send packets; allocating blocks of time as required by the *transmission* layer modules in order to allow them to both send and receive data; and switching the radio on/off as appropriate for the individual protocol.

A block of time is simply an interval during which the radio is exclusively handed over to a particular transmission module which has previously requested that the λMAC layer give it *n* milliseconds in order to send a packet sequence; conversely time blocks are also allocated when a packet comes in informing the local node that another node will be performing a packet sequence for a short period from now and so the local node should not give the radio over to other transmission layer requests for time. Note that when we talk about the good time to send a packet, we imply that this is a time with a high probability that the destination node will be able to receive the packet, which is information that the λMAC layer needs to keep track of as part of its time management role.

2.6 The λMAC framework

Given our new formulation of how a MAC protocol stack could be built, we can now define the required modules and connections for our new MAC stack (see Figure 2.2 for a pictorial overview of how these interact), which we refer to as the the λMAC framework.

- Packet layer - responsible for the actual sending/receiving of a packet, radio state changes (Rx/Tx/sleep) and for providing carrier sense functions (for CSMA-based λMAC protocols). The sending/receiving radio state here is “dumb” - it does things right now, with no options for delay or smart decisions considered. In the

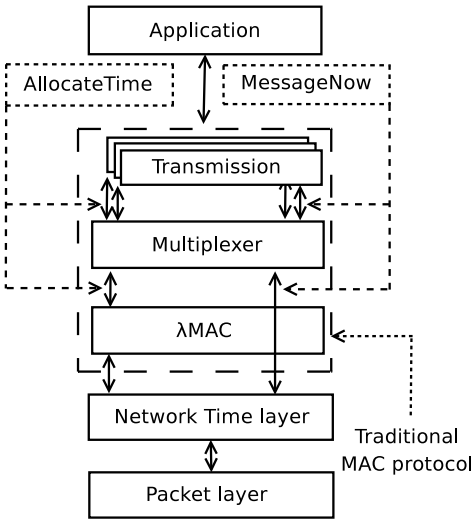


Figure 2.2: λMAC protocol stack

case of byte-based radios, we also provide a platform-specific byte-interface layer (which can only be talked to via the Packet layer), and for packet-based radios the Packet layer is a slim layer on top of the existing hardware capabilities. This allows us to abstract away from the differences of these two paradigms, as only packet-level information is required for the higher levels of the λMAC implementation.

- Network Time layer - responsible for determining and storing the local estimate of the current network time value in order to provide cross-network event synchronisation. This is not required by all λMAC layers, but given that network time information is useful to a large quantity of WSN MAC layers (due to the energy savings that can be made if nodes are able to agree when transmit/receive periods should be), that the information is potentially useful to other layers, and doing accurate timing information above the MAC layer is very difficult (given the uncertainty in send times of at least the 10-msec range created by most WSN MAC protocols, which may increase to 100s of msec for TDMA protocols), we implemented the Network Time layer here as a general service to the entire application stack.

  Responsibility for when to send packets is still the province of the λMAC layer, but the Network Time layer will add its own information on sending. The Network Time layer will also override the λMAC layer's decisions on when to stay awake on a periodic basis in order to do neighbour discovery. The overrides will make the radio be in receive mode more than it would be normally off, but will not switch the radio off when the MAC wishes it to be on, or switch the radio from transmit

to receive mode (or vice versa). The Network Time layer here provides the same interfaces as the Packet layer in addition to the Network Time interface in order to allow altering of packets (for the purposes of timing information) on their way to/from the Packet layer itself. For more information, see Section 2.6.2.

- λMAC - responsible for time management. Allocates time blocks in response to requests from the Transmission layer, at times that are considered to be "good". Talks to the Network Time layer in order to send its own control packets (if required), as well as for carrier sense checking in order to determine if the radio medium is free for sending (for CSMA-based λMAC layers), and decides when to switch the radio on and off. Passes packet send requests/receive events from/to the Transmission layer to/from the Network Time layer, possibly altering said packets along the way. Given the roles now allocated to other layers, the λMAC layer will be considerably smaller than a traditional MAC layer.

- Multiplexer - (de-)multiplexer to allow for the λMAC to only provide a single interface yet talk to many Transmission layer modules. This removes yet more common complexity from the λMAC, in accordance with our design goals.

- Transmission layer - contains the Unicast, Broadcast and other application-level primitives of this nature. Requests time blocks from the λMAC layer as required, and then sends packets during the allocated time. The transmission layer is fully explored in Section 2.7.

There is one limitation on the choice of MAC protocol for the λMAC layer - that it must be possible to allocate contiguous blocks of time that can be used for both sending and receiving by a node. This is possible for all contention-based MACs, and for some TDMA-based MACs, but this may require some alterations to the protocols.

2.6.1 λ interfaces

As we wish to define common connections between the λMAC and Transmission layers to enable reuse of the Transmission modules, we need to define some standard interfaces for these connections. We use here the terminology of nesC [37] to provide common semantics, and also because our reference implementation is implemented on top of TinyOS [49] (which is itself implemented in nesC). There should however be no obstructions to implementing this with any other WSN software platform.

We define two separate interfaces, AllocateTime (Table 2.1) and MessageNow (Table 2.3). AllocateTime defines the necessary functionality for a Transmission module to allocate time from the λMAC layer, and MessageNow allows the sending and receiving of messages during the allocated time. In general, a Transmission level module requires a single instance of the AllocateTime interface, plus one instance of the MessageNow interface per message type (e.g. the Broadcast module requires a single MessageNow, and a standard Unicast requires 4 MessageNow interfaces (RTS, CTS, DATA and ACK)). The λMAC layer, however, only needs to provide a single instance of each of AllocateTime

Name	Type	Args	Return	Function
requestBlock	command	uint16_t msec	result_t	Request an AllocateTime period of *msec* milliseconds. A return value of FAIL indicates a persistent failure i.e. the requested period is too long.
requestSafeBlock	command	uint16_t msec	result_t	Same as requestBlock, but should only be called after a previous AllocateTime period has run to completion, but has completely failed e.g. no response has been received from any other nodes at all.
startBlock	event		void	Called on the successful start of an AllocateTime period. Always corresponds to the last call to requestBlock.
sleepRemaining	command		void	Switch the radio off for the remaining length of the AllocateTime period. This is intended for periods when there will be packets in the air, but none of them are destined for this node.
sendTime	command	uint16_t length	uint16_t	Query how long a packet of *length* bytes should take to be transmitted with the relevant headers
endBlock	event		void	Called at the end of an AllocateTime period
notifyEndBlock	command		void	Notify module on end of period. endBlock events happen by default for locally initiated periods (periods starting with a startBlock()), but are switched off by default for non-locally initiated periods. notifyEndBlock() switches on endBlock events for the currently active AllocateTime period.

Table 2.1: AllocateTime interface

Name	Type	Args	Return	Function
phyRequired	event		void	Indicates that a packet (any packet) should be sent as soon as possible by the λMAC layer in order to maintain time synchronisation. See Section 2.6.2.2.

Table 2.2: PhyRequired interface

Name	Type	Args	Return	Function
send	command	TOS_MsgPtr msg, uint8_t length	result_t	Sends a packet right now. Fails if we are already sending something. Should only be called during an AllocateTime period.
sendDone	event	TOS_MsgPtr msg	void	Called on completion of a send()
setAddressFiltering	command	bool enable	void	Enables/Disables automatic destination address filtering for this interface i.e. dropping all incoming packets not destined either for this node or for the broadcast address. Default is not to filter. If filtering is switched on, packets not destined for this node will cause sleepRemaining() to be called in order to avoid overhearing the packet sequence.
receive	event	TOS_MsgPtr msg, uint16_t fromAddr	bool	Called when a message comes in that is not filtered (see setAddressFiltering). Implementations should return TRUE if they wish to stay awake for the rest of the AllocateTime period, and FALSE otherwise.
reservedBytes	command		uint8_t	Number of bytes reserved at the beginning of the data section of the TOS_Msg by lower layers
setPreambleLength	command	uint8_t length	void	Set length of packet preamble to *length* bytes. Defaults to 1 if not called.

Table 2.3: MessageNow interface

Name	Type	Args	Return	Function
setFrameTime	command	uint32_t msec, fuzz_t fuzz	void	Set time between frame timers (*msec* milliseconds) as well as allowable fuzz time (LOW/HIGH_FUZZ)
frameIndex	command		uint32_t	Determine location within the current frame i.e. milliseconds since last frame timer.
networkTime	async command	networktime_t *temp	void	Get a copy of the current local value of the Network Timer. May or may not be currently synchronised with other nodes.
frame	async event		void	Frame Timer event. Fired only when this node is synchronised to the other nodes in the network (or has waited long enough to confirm that there are probably no other synchronised nodes in the local neighbourhood).
frameGuaranteed	async event		void	Fired when all nodes should now have received frame events (due to limits on their desynchronisation set by the *fuzz value).*
frameSkipped	async event		void	Indicates that one or more frame() events have been skipped due to Network Timer alterations.

Table 2.4: TimeSync interface

and MessageNow to the Multiplexer module. The Multiplexer module provides generic multiplexing services to create a parametrised interface to both AllocateTime and MessageNow, thus enabling the capability for multiple Transmission layer modules to be enabled in a single application, without having to deal with the multiplexing complexity in each λMAC layer.

Individual Transmission layer modules could be implemented using a single MessageNow interface per module. However for modules that require multiple message types (e.g. Unicast), the implementers of the Transmission modules would have to both add their own type field to the sent messages, and do de-multiplexing of the different types at the receiver side. As the Multiplexer module allows for multiple instances of MessageNow already (in order to allow multiple Transmission modules in a single application), the Transmission layer protocol design can be simplified by using multiple MessageNow interfaces, and this also removes the necessity for the overhead of an additional type field.

The interface between the packet layer and the λMAC layer is much simpler, and as this is more in keeping with traditional WSN MAC design, we will not cover it in detail here. The Packet layer must provide interfaces to change the radio state (Tx/Rx/sleep), and also to send/receive packets - similar to the send/sendDone/receive commands and events of MessageNow. For a CSMA-based λMAC layer, the Packet layer will also require an interface to carrier sense operations. As we stated before, the Packet layer is "dumb" - all of the smart decisions regarding when to send, to listen and to sleep are decided by the particular λMAC layer in use.

2.6.2 Network Time

In order for many MAC protocols to operate correctly, they require a mechanism to synchronise nodes so that differing nodes can agree on events happening at the same time e.g. synchronised awake times. Additionally, placing this within the packet layer also allows integrating time synchronisation information into each outgoing packet, thus reducing the need for additional control packets whenever data packets are being sent. However, as we wish the Network Time layer to not override λMAC-layer decisions about when to send packets, in the case where a node does not have a sufficient rate of outgoing packets to guarantee time synchronisation, the Network Time layer will send a phyRequired event (Table 2.2) to the λMAC layer requesting that it send a packet "soon" in order to maintain time synchronisation.

In keeping with the idea of the Network Time layer as a generic layer, and also because we wish to provide information to modules other than the λMAC layer, we need to define the timing information appropriately. We started with the work of Li et al [73] on the *global schedule algorithm* (GSA), but then expanded it one step further. In GSA, nodes keep track of a local *age* parameter, intialised to zero, which is updated to represent how much time has passed, and add this information to their outgoing packets. Initially, the *age* parameter represents how long a particular node has been switched on, but if a node sees an incoming packet with a greater *age* than the local *age*, the local *age* is

updated to be the same as the incoming packet, thus allowing the network to converge towards a shared timing value based on the oldest (first switched-on) node's *age*.

In the original implementation of GSA, schedule information (time since last frame timer) was also distributed with the *age* value in order to calculate the correct current frame timer for the MAC protocol. In the λMAC framework, we have a separate TimeSync module, which is used by the λMAC framework as a storage location for the current local value of the *age* value. However, TimeSync provides periodic "frame timers" (of variable length up to $(2^{32}-1)$ms) to all application modules that require this capability (not just λMAC layers that need it) - e.g. for experiments that require an entire field of nodes to make a measurement at the same time (a commonly wanted requirement for many biological experiments being proposed for sensor networks). We do this by using the *age* value modulus the frame length to provide a frame timer every time *(localAge mod FrameTime) = 0*. This allows the creation of multiple frame timers for different application modules, while only requiring synchronisation on the single *age* value.

The time synchronisation implemented within the Network Time layer has some relation to the more general field of time synchronisation in distributed systems (e.g. NTP [84]), but bears closer resemblance to Lamport logical clocks [69], in that we are more concerned about agreement within the network on a value at any given time, as opposed to synchronising with external clocks i.e. "wall clock" time. However, the mechanism for incrementing an individual node's opinion of the current value for the global clock is based upon "real" clock time, and so we can maintain relatively tight synchronisation between nodes without requiring continuous exchanges of time data.

The current specification of the Network Time limits synchronisation granularity to 1 millisecond, but that could be expanded in the future. The current limit of 1ms is a trade-off due to limits of the current primary hardware platform (specifically, the ATMega128 [7] processor, and its limitation of only having an 8-bit timer active when in "sleep" mode, which at 1ms granularity requires the processor to wakeup every 256ms to handle overflow events in the timer), but future work may be able to reduce the granularity to allow for tighter time synchronisation.

2.6.2.1 Fuzz values

All of the periodic frame timers also have an allowable "fuzz" value - if because of updating the local clock, we jump over the time when we should have fired a frame timer, but we jump over by less than the "fuzz" value, then we fire the timer anyways. This bounds the acceptable jitter in the frame timer event. In the event we jump too far over the event point, the safest approach is usually just to skip the event entirely and wait for the next one (e.g. not doing an awake period for a TDMA protocol that is drastically out of sync with other nodes). This allows us to cope with small changes in the network clock due to varying speeds of clocks on different nodes. We implement this using a frameSkipped() event to signal skipping of events, and a frameGuaranteed() event that is fired "fuzz" milliseconds after the frame() event, which has the guarantee

that all synchronised nodes should have received their frame() event by the time a node gets a frameGuaranteed() event.

Earlier implementations of the timer mechanism gave the fuzz value as an absolute number of milliseconds, but after this was used in several different protocols we concluded that this did not fit well with the use cases that we were seeing. We in fact only saw two cases for values of fuzz - for MAC protocols, with a very low fuzz value and frame sizes in the typically <2 second range (sometimes as low as 50 milliseconds); and for other protocols (notably aggregation) that had much larger frames (10s of seconds to minutes or greater), and high fuzz values (>1 second).

The two cases were optimising for different constraints, but this was not fully reflected in their fuzz values. In the MAC case, the emphasis was on a fuzz value as low as possible (given hardware constraints). MAC protocols could cope with an occasional frameSkipped() event if the fuzz value was set very low, as another event will arrive shortly afterwards, but the emphasis was on tight synchronisation between frame events on separate nodes. Typical fuzz values for MAC protocols were 2-4 milliseconds, depending on the whims of the author of the protocol. Note that these numbers were arbitrary and picked according to individual guesswork about how tight the synchronisation between nodes was likely to be on a particular hardware platform (and is also related to the overall current 1ms granularity of the Network Time values). Also, too high fuzz values for MAC protocols will effect their performance significantly, as the fuzz value determines the separation between frame() and frameGuaranteed(), and a number of protocols will switch the radio in receive mode for that period. Ergo, increases in the fuzz value would increase this time, and so keeping it as small as possible will reduce power usage.

For protocols with much longer frames, a frameSkipped() is often quite bad, as it will result in much larger gap between frame() events v.s. protocols with short frames. Additionally, for most long frame protocols, tight synchronisation is not required, as they are generally higher up the stack than MAC protocols, and so will be subject to other semi-random delays (e.g. send delays from MAC or routing protocols) even if they had tight synchronisation. Often, the goal is just "reasonable" synchronisation, and the fuzz values were again often arbitrarily picked numbers with little thought in their decision process.

We therefore limited the fuzz values to two values marked as LOW_FUZZ and HIGH_FUZZ. LOW_FUZZ will always be a value as low as possible (given platform-specific knowledge outside the domain of the users of the frame timers), optimising for tight synchronisation. HIGH_FUZZ will be a value optimising for no frameSkipped() events, with current values being approximately 5% of the frame length specified by the user. Taking the decision for the actual values out of the hands of the users, and instead using a more abstract value with clearer semantics regarding what the user actually wants removes most of the problems we saw here, as well as being in line with the design intentions of the λMAC framework by providing code paths (platform-specific decisions about good fuzz values) that would otherwise be duplicated in many MAC protocols.

2.6.2.2 Maintaining synchronisation

The most important aspect of maintaining time synchronisation is establishing synchronisation in the first place i.e. discovering neighbour nodes to synchronise with. In the Network Time layer, we use the concept of a "sync period" to achieve this. Initially, when a node first starts up, it stays awake for an entire sync period (currently set to 7s) in order to discover other nodes to synchronise to. Additionally, the Network Time layer will periodically do neighbour discovery every *n* sync periods. Our current implementation does adaptive neighbour discovery, searching every 3 periods if no neighbours have been discovered or every 30 periods if neighbours have been found. This translates to 21s or 210s (based on the 7s sync period) depending on whether other nodes have been found. One implication of the sync period length is that a packet must be transmitted by every node at least once per sync period (with a PhyRequired event being fired by the Network Time layer if a node is not sending enough packets at a particular time), which causes some level of overhead. Increasing the sync period length would reduce the number of overhead messages during times when the node does not need to send packets for other purposes, but would also increase startup times, as nodes need to stay awake for a complete sync period on startup. Further investigation is still being done into better values for the constants mentioned here, but the best values will always be application dependant.

One of the potential problematic cases for MAC time synchronisation is how to handle situations where two different groups of nodes with different values for the network time come into contact with each other. For some other time synchronisation mechanisms (e.g. S-MAC's scheduling algorithms [148]) attempts are made either to create a "merged" time value from the two different network time values, or sometimes to maintain multiple different reference values for the time synchronisation mechanism at the same time.

The Network Time layer does neither of these, as both scenarios tend to lead to overly complex situations (especially once more than two groups of separate nodes come into contact). Instead, the existing synchronisation method is used i.e. the oldest known age is still the goal for synchronisation. For the two groups coming into contact - Groups A and B, who have respectively an older and a newer network time value - the actions will be as follows. Group A (older) will actually do nothing - their network time values will remain "static" (updating purely due to elapsed real time, but no drastic changes). Group B, as the nodes within it receive messages from Group A (either via accidental overhearing, or during sync periods), will effectively become "absorbed" by Group A, because the Group B nodes will update their network time values to be in synchronisation with Group A. This may cause frameSkipped events (see Section 2.6.2.1) due to the incrementing of the timers of Group B nodes, but the disruption should be short lived for any given node.

One of the major reasons behind the relative stability of the Network Time layer is the use of the "oldest" value as a synchronisation goal, in combination with choosing the size of the local storage variable for the Network Time value to be large enough to avoid wrapping around within the feasable lifetimes of most sensor networks (34 years with the current implementation of 40-bit timers). Even if large quantities of nodes are repeatedly

reset (including the "oldest" node) the "oldest" value will tend to be maintained within the network, provided at least a single node still retains the value. Part of the logic behind this choice is that new "oldest" nodes are likely to appear infrequently in a network (as node failures should not cause the creation of "older" nodes), and most nodes will only have to do significant "jumps" in their network time values early on during their lifetimes. As jumps are disruptive to any processes relying on the Network Time value, reducing the number of jumps should be a priority for a Network Time system.

2.6.2.3 Criticism

One possible criticism of the Network Time layer, when contrasted to the choices of timing mechanisms in earlier MAC protocol designs, is that it attempts to generate "network" time synchronisation, when most MACs are quite happy with merely local (1-hop neighbours only in many cases) synchronisation. Firstly, given that time synchronisation is difficult to achieve above the MAC protocol level, and it is a useful primitive to other layers (which would not need to do additional synchronisation when using the Network Time system), doing network synchronisation at this level is a good idea.

Secondly, as many MACs need local synchronisation, we submit that our generic network time sync mechanism is not actually much more complicated (either in terms of messaging or processing overhead) than most time sync mechanisms designed for a specific MAC; indeed, it is simpler than some MAC protocol sync mechanisms [22], while providing the required level of synchronisation for those protocols.

Thirdly, our anecdotal experiences while speaking to other MAC designers, combined with our experience of doing this ourselves, is that doing time synchronisation correctly on real nodes is *hard*, and it tends to dissuade many budding protocol builders, when their effort should be focused on the things that make their MAC protocol different, not the common features that they should have provided for them.

We conclude that given all of these issues, the creation of a general-purpose time synchronisation layer and providing it as an automatic service of our framework is a good choice for most MAC protocols.

2.7 Transmission layer modules

In this section we will look at how to implement Transmission layer modules, with a focus towards the standard set of WSN Transmission modules on top of the λMAC layers, i.e. the set of functions that would be expected from a standard MAC protocol. An exploration of what can be done with non-standard modules is in Section 2.12.

2.7.1 Notes on Transmission module design

Before we go into a more detailed look at how to build basic Transmission modules, a number of features of the MessageNow and AllocateTime interfaces should be noted:

- The point of an AllocateTime period is to acquire blocks of time in order to send packets with a reasonable guarantee about a node's neighbours being in a state where they are able to receive packets. A node does not need to be in an AllocateTime period for any other purpose.
- The AllocateTime period (as marked by a startBlock() event) is only started when a certain level of guarantee can be given that the radio medium will be at least relatively quiet. In CSMA-based protocols this will be done via a carrier sense mechanism of random length (to resolve contention issues between multiple nodes wishing to start AllocateTime), and in TDMA-based protocols this is guaranteed by the time slot mechanisms.
- The λMAC layer will piggyback information about the remaining AllocateTime period on outgoing packets, in order to place other nodes into the AllocateTime state as well.
- Once an AllocateTime period is started, it cannot be stopped. This is because of the difficulty of telling other (possibly asleep) nodes of this change of plans. A node can be told to go to sleep for the rest of the time period however (via sleepRemaining()).
- Setting setAddressFiltering() is recommended for all protocols that set the destination address to non-broadcast addresses, as this will enable the λMAC layer to reduce the level of calls to the Transmission layer, and will also simplify Transmission layer design by avoiding duplication of address filtering code. The λMAC layer will also be able to use the transmitted AllocateTime value to avoid overhearing the rest of this packet sequence.
- A receive() event's return value says whether to stay awake for the rest of this AllocateTime period or not. This is automatically handled using sleepRemaining(), and the Transmission layer will not generally need to call sleepRemaining except in certain special situations (for example, if you wish to receive packets for a short period after receive(), then go to sleep).

2.7.2 Broadcast

Broadcast is simply implemented on top of a single MessageNow and AllocateTime pair. Sending is implemented as follows

1. Call requestBlock() for sendTime(packet length) milliseconds
2. On startBlock(), call send().
3. On sendDone(), call sleepRemaining()

Receiving is also very simple, as all instances of receive() will return FALSE, as we will no longer be receiving additional packets during this period.

2.7.3 Unicast

Unicast is somewhat more complicated than Broadcast, partly because it can have variants both with and without RTS/CTS. For the case with RTS/CTS, an example implementation runs as follows. During the initialisation of this module, we should call setAddressFiltering() with TRUE, and set *control_length* to the return value of sendTime(0), as this is the length of a control (RTS, CTS or ACK) packet, because they contain no data, only MAC headers.

To send a packet, we first calculate *packet_time* as sendTime(packet length) + 3**control_length* plus some platform-dependant allowance for processing and radio state transition delays. We need 3 *control_length* intervals for the RTS, CTS and ACK packets. We then call requestBlock() with *packet_time*. On startBlock() (as we have a reasonable guarantee about the time slot, so we can start immediately), we start to cascade through the RTS-CTS-DATA-ACK sequence i.e. we send an RTS packet using send(), wait to receive a CTS, then send a DATA packet with send(), then wait to receive the ACK. We return FALSE from the ACK receive in order to sleep for any left over processing time.

At the destination receiver node, we first see a receive() with an RTS packet. As this is destined for us, we return TRUE from receive(), after first posting a task to send a CTS with send(). Then, the receiver waits for DATA, sends an ACK with send() and calls sleepRemaining() (in order to go to sleep for any remaining left over processing time). Other nodes that are not the destination for this Unicast sequence will automatically filter out these messages and go to sleep (due to the use of setAddressFiltering()).

This is a simplified description for an example Unicast module, and our complete implementation includes retries for lost/missed packets. However, it gives a flavour of how Unicast can be implemented on top of the λMAC layer.

2.8 Integrating existing MAC types

Now that we have shown how we intend to split up existing monolithic MAC protocols into a more generic and reusable stack (Section 2.6), and described how that stack works (Sections 2.6.1, 2.6.2 and 2.7), we need to go back and show that all of this can work with existing MAC protocols.

We divide WSN MAC protocols into 3 groups as shown in Figure 2.3. We initially divide protocols into send timing allocated according to a "good" time for the sender v.s. "good" time for the receiver - this separates out protocols like Crankshaft [42] (which allocates when

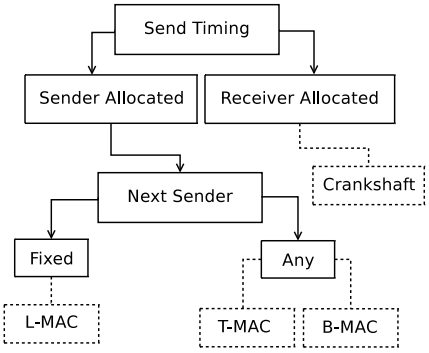


Figure 2.3: WSN MAC protocol division

to send a packet depending on the destination address of a packet) from the majority of other protocols. To our knowledge, only Crankshaft and PMAC [154] use this scheme. We can then further subdivide sender-allocated protocols into fixed v.s. any next sender node protocols. This is approximately the division between TDMA- and CSMA-based protocols, with T-MAC and B-MAC [99] both being in the latter group (despite the differences between the two) and LMAC being an example of TDMA.

In the next two sections we intend to describe our implementations of a λMAC implementation of T-MAC (Section 2.9) and LMAC (Section 2.10). We will go into further details of the protocol implementations in the relevant sections, but given the built-in concept of time allocation due to the scheduler mechanisms in each MAC protocol, conversion to the λMAC framework was relatively simple. λLMAC caused more difficulties due to the single-sender semantics of TDMA time allocation, but as we will show, was still feasible with some minor modifications.

B-MAC [99] is a prominent example of a different type of protocol, despite our grouping it with T-MAC - no consistent scheduling, continual sampling of the radio medium (using LPL in B-MAC's case), and a complete lack of built-in time management. One of the challenges for the λMAC framework was to be sufficiently flexible to be capable of implementing such a protocol, while still providing the same level of functionality as with other MAC protocols. However, despite the differences to other protocols, most of the issues that we would expect to encounter during the implementation of λB-MAC have already been dealt with during our creation of λT-MAC (which follows from our grouping of the protocols together). Implementing B-MAC given our work on T-MAC requires two significant blocks of new code - the LPL channel sampling can be implemented like the active/sleep periods of T-MAC (see Section 2.9.1), except much shorter and with a fixed awake time rather than T-MAC's dynamic one; and use of the setPreambleLength() function of the MessageNow interface (see Table 2.3) is needed to allow for the longer preambles required by LPL.

We believe that by showing that T-MAC, LMAC and B-MAC can be implemented with the λMAC framework, and by providing data from our experiments running the first two on our testbed, we adequately demonstrate that the λMAC framework is suitably generic to be able to be a base for implementing the majority of sender allocated MAC protocols. The receiver allocated group is a significantly smaller subsection of the entire range of current MAC protocols, but as the reference implementation of Crankshaft is based upon our work here, we do not believe the differences are irreconcilable. In Section 2.14.1, we outline some possible extensions to the λMAC Framework that would both fully enable receiver allocated protocols, and reduce the level of effort required for implementing TDMA protocols.

2.9 λT-MAC

So far we have mostly looked at generic concepts of a λMAC layer. In this section, we describe our implementation of the λT-MAC layer, based on T-MAC [24] for TinyOS [49].

2.9.1 Scheduling

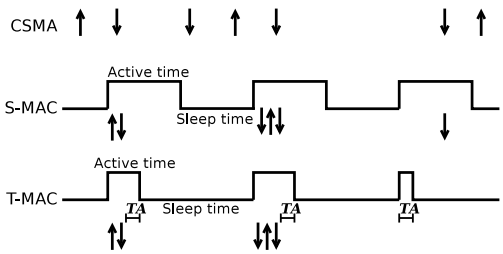


Figure 2.4: Scheduling with different MACs

T-MAC is a CSMA-based MAC protocol, derived from S-MAC [148], but with adaptive duty cycling. The adaptive duty cycling is based on the idea of going to sleep shortly (*TA* milliseconds, defined by the time needed to receive a minimal packet, process it, and send another minimal packet) after the last "interesting" event - which can be a message going out, another message coming in or the periodic firing of a frame timer every so often (see Figure 2.4). The frame timer length is a trade off between energy efficiency (with longer sleep times between awake periods) and latency (due to the length of sleep before the next time we can send a packet).

We took the implementation of T-MAC for TinyOS, and adapted it to provide a λMAC layer, including the removal of its integrated Broadcast and Unicast functionality. Adapting the existing T-MAC protocol to provide the λMAC functionality was relatively simple. We used the frame timers from the Network Time layer to remove a lot of the complexity from T-MAC (68% smaller code base, see Section 2.11.1), including the removal of a significant part of the existing code which was dedicated to schedule synchronisation (including discovery of new schedules); a role now subsumed by the Network Time layer (see Section 2.6.2.2). On a requestBlock() call, λT-MAC places the requested amount of AllocateTime into a nextAllocateTime variable. When T-MAC would normally check if it has a packet to send, λT-MAC instead checks if nextAllocateTime is not 0, and if so requests that the packet layer do a carrier sense check. If the carrier sense returns an idle radio medium, then startBlock() is called and λT-MAC waits until the end of the AllocateTime period before doing anything else. MessageNow send() and receive()'s pass almost uninhibited through the λT-MAC layer. Notably, the send() is not delayed waiting for anything else to complete, but is passed through to the packet layer as rapidly as possible. If we get a phyRequired event (a request from the Network Time layer for a packet to be sent), λT-MAC sends out a Sync packet - a packet with no actual data payload, and only containing timing information in order to maintain the inter-node time synchronisation.

2.9.2 Testbed data

In order to test whether the λMAC concept was viable, we compared λT-MAC to the existing T-MAC implementation. Our testing was done on the TNOde platform, a WSN node derived from the mica2dot design [48]. We wished to check whether the switching from a monolithic MAC protocol to the separated λMAC design had affected the compiled program size for a complete program (including λT-MAC, the λMAC framework, a simple test program and the standard TinyOS system code), maximum packet transmission rate and awake/sleep ratios.

	ROM Size	RAM Size	Maximum packet rate
T-MAC	22518	2123	9.9 packets/second
λT-MAC	21678	2192	9.3 packets/second

Table 2.5: Continual sending test

	ROM Size	RAM Size	Duty cycle at 1 packet/second
T-MAC	22726	2133	14.3%
λT-MAC	21798	2202	14.4%

Table 2.6: Broadcast cycle time test

For our application testing, one of the example T-MAC applications was used - a simple radio testing application. All packets in the test application had 10 bytes of dummy data in them, and all experiments were run for 60 seconds. We compiled the application with nesC 1.2.4 and gcc 4.0.2 for the AVR.

To test the maximum output packet rate, we used a version of the application that sends broadcast packets continually. Notably, T-MAC was not designed as a high data rate MAC, but we felt this was still a useful reference test checking for similar performance between the two implementations. The result of this test are in Table 2.5, and show a reduction of the packet rate of only 6%, which for an unoptimised reference λMAC was we felt was an acceptable loss.

We also tested the active duty cycle of the protocols while sending 1 test packet every second. The result of this test are in Table 2.6, which shows that the change to the λT-MAC implementation resulted in a $<$1% increase in the amount of time that the node needed to stay awake in order to send the requested packets. The ˜14% duty cycle is quite high for T-MAC, but this is due to a combination of a 610ms frame timer and a 69ms *TA*, giving a minimum duty cycle of ˜11% even without any packets being sent, and optimisation of the core protocol implementation could improve this significantly.

Note that for both variations of the test application (Section 2.9.2) that the compiled ROM size when using λT-MAC was reduced by approximately 840 bytes v.s. using T-MAC (the exact reduction varies, depending on the level of optimisation that the compiler was able to do for the particular application).

2.10 λLMAC

LMAC [50] is a TDMA-based MAC protocol, aimed at giving WSN nodes the opportunity to communicate collision-free, and at minimising the overhead of the physical layer by reducing the number of transceiver state changes. The MAC protocol is self-organising in terms of time slot assignment and synchronisation, starting from a sink node (specified by the application). Upon start-up, the sink node sets a frame schedule

and chooses the first slot in the frame as its sending slot. Next, one-hop neighbours receiving the sink's transmissions, choose their sending slots based on the frame schedule of the sink node. This is then repeated for all next-hop neighbours. When an application wants to send a message, LMAC delays the transmission until the start of the node's next sending slot.

2.10.1 Implementation

We created a TinyOS implementation of λLMAC based on the protocol description and the OMNeT++ [138] code available from the LMAC authors. For time synchronisation between the nodes, we used the Network Time layer, and so were able to use a frame timer to determine the start of each slot. This way, all nodes agree on the start time of all slots. When using a frame timer to determine only the start of each LMAC frame, intermediate clock updates during the frame may lead to inaccurate start times of slots near the end of an LMAC frame.

Although λMAC supports sending multiple packets in a single slot, in LMAC it is only possible for a node to transmit a single message per frame. The authors suggest gluing together multiple messages to the same destination to prevent high latency, but this suggestion is not implemented in the available OMNeT++ program code. To make our results comparable to the OMNeT++ implementation we had available, we did not implement this feature either.

On a requestBlock() call, λLMAC sets a flag indicating that there is a packet waiting to be sent at the node's next time slot. During its time slot, a node will always transmit a packet. If a node has no data to send, an empty Sync packet is sent to keep the network synchronised. Otherwise λLMAC calls startBlock() and waits until the end of the time slot to call endBlock().

Since a TDMA-based MAC-protocol does not need the full Unicast RTS/CTS/DATA/ACK sequence to keep other nodes from transmitting at the same time, we created a Unicast module that only sends the DATA packet. As the TinyOS message header already contains information about destination node and packet length, this information was removed from the LMAC-specific header.

2.11 Testing

We performed a series of tests comparing the λMAC versions of LMAC and T-MAC to earlier 'monolithic' implementations. In the case of T-MAC, we had the existing implementation for TinyOS to compare against. As there was no existing TinyOS code for LMAC, we had to work from simulation data. Our simulation work is based upon the simulation framework from [71], with various parameters (byte times, frame times, etc) altered in line with the parameters used by the λLMAC implementation.

We used two tests: a Unicast test (Figure 2.5), with all nodes sending to a single 'sink' node; and a 'Cloud' test (Figure 2.6), with two nodes designated as A and B trying to send packets to each other, while the other nodes send broadcast data around them. In the

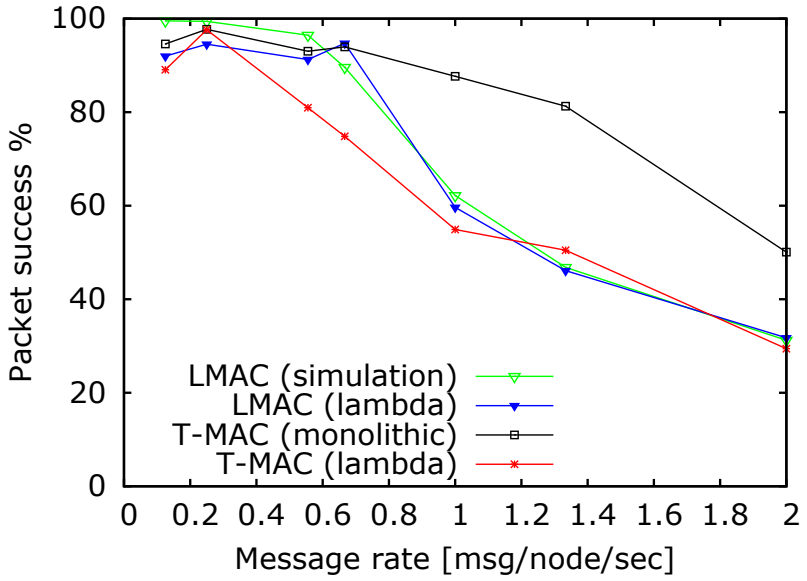


Figure 2.5: Unicast test

case of the Cloud test, we measure the packet success rate as the success rate for packets between A and B, ignoring all other packets. The testbed data is from our deployed network of 24 TNOdes, with power levels set to create a single-cell network with all 24 nodes within one hop of each other; the simulation data is also from a single-cell network with 24 nodes. LMAC was set to a slot time of 50ms, with 32 slots, giving a 1.6s frame. T-MAC was set to the standard frame time of 610ms in all cases.

As can be expected from this form of multi-environment experiment, we encountered a number of rather unexpected results; however, the end data does show a number of useful things. The Unicast test showed remarkably similar numbers for both of the LMAC implementations - we expected the drop-off curve illustrated on the graph as we start to exceed the 1 packet/frame limits of LMAC. T-MAC, on the other hand, shows a significant difference in the data. Both versions of T-MAC illustrate the characteristic curve of an overloaded network, but λT-MAC appears to be suffering from additional factors reducing its capability to transmit and receive packets successfully. As the packet sizes are relatively unchanged between implementations, and they both require the same amount of sync packets in order to maintain time consistency, we are currently unsure as to the cause of this drop. However, noting the good data from LMAC, we suspect the issue remains in our λT-MAC layer rather than the λMAC framework.

The Cloud test was designed as an example of a test that LMAC should succeed at, as illustrated by the near-perfect line of the simulation LMAC. One current issue with the simulation environment is its lack of detail regarding the quality of radio links, and

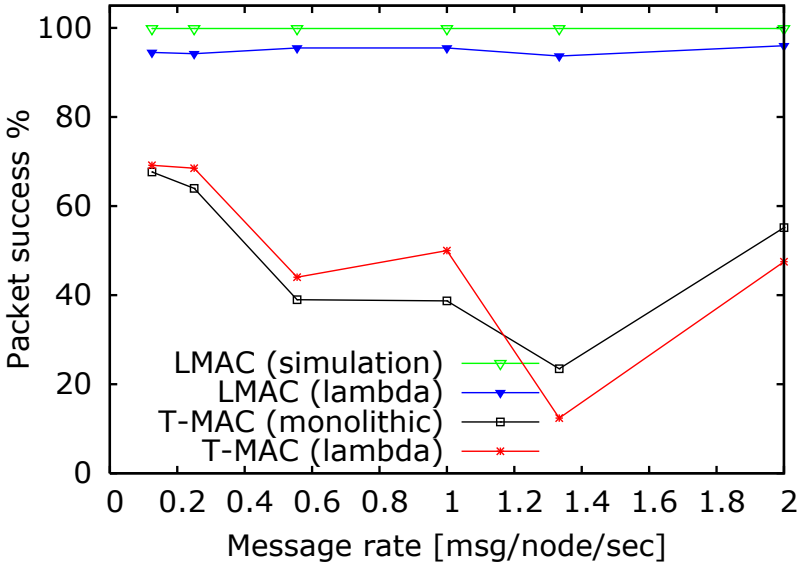


Figure 2.6: 'Cloud' test

this is is probably why λLMAC is unable to sustain data rates at this level. λT-MAC on the other hand, outperforms monolithic T-MAC on this test, showing that the earlier performance drop does not necessarily hold for all application scenarios. This result, in combination with the results from Section 2.9.2 further shows that the problems λT-MAC encountered in the Unicast test will not occur in all application scenarios, giving greater confidence in its general applicability.

2.11.1 Code Size

To check how large the implementations of the core modules were in each case, we measured the nesC code with SLOCCount [140] (Source Lines Of Code). λT-MAC and λLMAC's proportion of the total stack is in Table 2.7. Note that this is lines of code for the λMAC layers only, as opposed to the earlier data in Section 2.9.2 regarding compiled size for a complete application.

Component	Lines of Code	% of MAC Stack
MAC Framework	3961	Variable
λT-MAC	1426	26%
λLMAC	814	17%

Table 2.7: λMAC sizes

For λT-MAC, we had an existing TinyOS implementation, and so we could compare λT-MAC to the older implementation. The original "monolithic" T-MAC had a total of 4367 lines of code v.s. the 1426 lines of λT-MAC, making λT-MAC only 32% of the original size. Notably, we do not count the lines of code in the MAC framework itself that are required by λT-MAC, as we only count the code that would have to be written by someone building a new implementation of the MAC protocol in each case, which is the point of the code reuse due to the MAC framework.

2.11.2 Power tests

To further check the performance of λT-MAC, we wanted to measure its power usage. Unfortunately, the existing TinyOS T-MAC implementation turned out to be not switching off the radio as much as it we would expect, causing uneven power data and hindering direct comparisons. Therefore we decided to stick to scenarios where existing research (i.e. the original T-MAC paper [24]) provided us with examples of how a T-MAC implementation should behave in terms of power used. We used a simple two-node, unicast sender-receiver pair, with the sender node transmitting 1 packet/second.

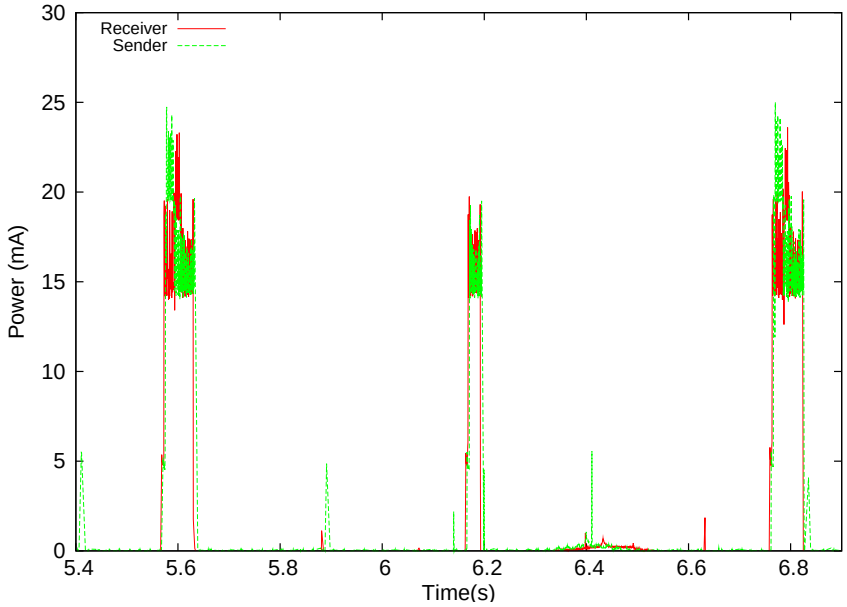


Figure 2.7: Basic λT-MAC power trace

Figure 2.7 shows ˜1.5 seconds of the power readings from this application, with λT-MAC demonstrating the classic T-MAC "awake for short time, sleep for long period" graph, clearly demonstrating good synchronisation between the two nodes.

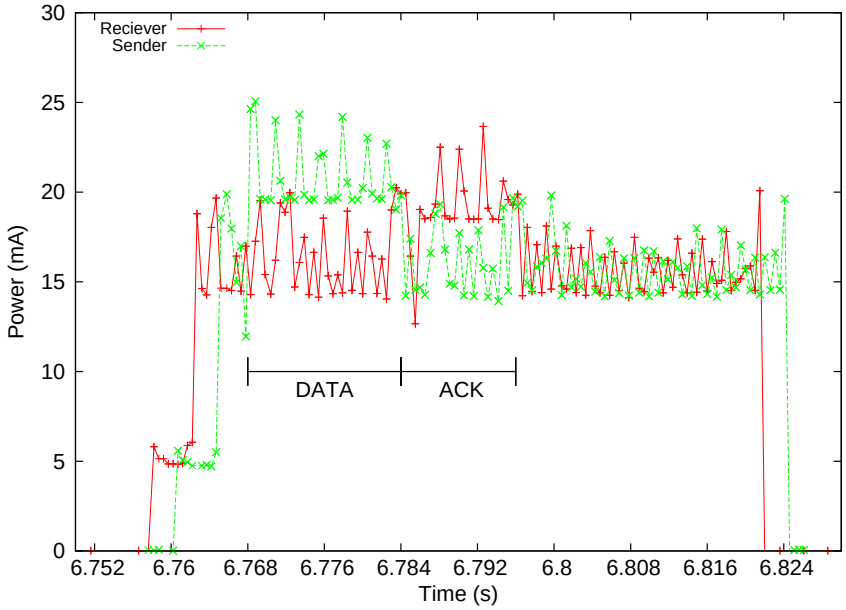


Figure 2.8: Detail of DATA/ACK sequence

Figure 2.8 shows a detail from part of a DATA/ACK sequence with the nodes. Between 6.768s and 6.784s the DATA packet is being transmitted, and the ACK is being transmitted between 6.784s and 6.796s.

The amount of power used, and the relative amounts of time spent in transmit and receive mode appears to be consistent with our expectations for a T-MAC implementation (see Figure 2.4 for the typical T-MAC power sequence), giving us additional confidence in the ability of the λMAC framework to correctly implement this protocol.

2.12 Further Transmission modules

In this section we look at some Transmission modules that can be implemented on top of the λMAC layer that would not be considered part of a standard MAC protocol, but would provide useful additional primitives for other applications. Notably, these would be non-trivial to add to most normal MAC protocols, as we would either have to try and build them out of Broadcast and Unicast operations, which would be significantly sub-optimal; or we would need to rebuild the MAC entirely. Our modular approach makes these additions not only possible, but relatively easy.

2.12.1 ExOR

ExOR (Extremely Optimistic Routing) is a "one send, many replies" approach to reliable multicast for routing protocols, first explored by Biswas and Morris [13], and an extended version is proposed by the Guesswork routing protocol ([2] and Chapter 3). Both variants can be implemented on top of the MessageNow and AllocateTime interfaces, but would require significant effort to implement inside existing MAC protocols.

An ExOR sending node sends a packet that not only contains the data for the packet, but also a list of other nodes that should respond (in the order that they are meant to respond in). Every node that is in the list that receives the packet waits sufficient time for all of the earlier nodes in the list to respond, and then sends an ACK to the sender node (see Figure 2.9). This can be used for a number of things - for example, implementing Reliable Broadcast, as the sending node knows that all nodes that it receives an ACK from have received the packet; or making a best-effort next-hop transfer in a routing algorithm (by using the ACKs to implement an election mechanism to pick the "best" possible next-hop node that has correctly received the original packet).

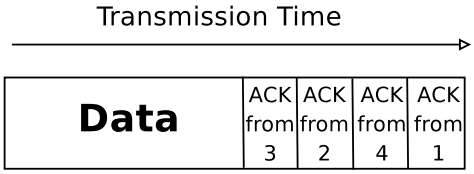


Figure 2.9: Example ExOR packet timeline

From the point of view of implementing ExOR as a Transmission layer, it can be considered as a variant of Unicast, with no RTS/CTS and a series of receiver nodes, all of which need to pause a variable amount of time before sending their ACK packets, and then call sleepRemaining() to avoid overhearing the remaining ACKs. As the destination address field is invalid in this case (as there is a list of destination nodes later on in the packet), we need to switch off address filtering (using setAddressFiltering()) and do the separation between destination and non-destination receiver nodes in the Transmission layer. We will revisit ExOR in Chapter 3, and show there how it can be built in more detail.

2.12.2 Priority Queueing

Another possibility that arises once the λMAC layer has been implemented is priority queueing [74, 132] which has been requested by various applications - namely, allowing for messages to be sent out in an order different from that which they were received (either from other nodes in routing scenarios, or events from local sensors). In standard MAC protocols, the "send" method is a fire-and-forget concept i.e. once the "send" has been called, cancelling the message (or even being aware of whether the message is queued or actually being sent right now) is impossible.

However, using the λMAC layer, a priority queue can be implemented. Specifically, that requestBlock() corresponds to the normal "send" call, and that although the corresponding startBlock() would normally be the time to send the original packet, any other packet can be sent. To implement a good priority queue, requestBlock() should be called when there is a packet to send out, but with a length appropriate to the maximum size packet that we may wish to send. On startBlock(), we then send the highest priority packet that we have on hand (which may well have arrived since the requestBlock() call), and call sleepRemaining() on sendDone() to trim the listening time appropriately to the length of the packet we actually sent.

2.13 Related work

At some levels, the core concepts of λMACs v.s. traditional MAC protocols can be viewed as similar to the micro v.s. macro-kernel debate in conventional operating systems. In common with microkernel design [34, 106], the λMAC layer is able to separate out parts of a WSN application that would normally be considered a very complex part of the system (as both MAC layers and operating system kernels in general tend to be regarded by many programmers as "here be dragons" areas of code), and these separated parts are then able to be altered with a significantly lower chance of affecting the rest of the codebase.

Polastre et. al [100] proposed the Sensornet Protocol (SP) that provided a greater level of control to applications wishing to influence the choices made by lower level protocols. Their system created a much more horizontal design for the various levels of an application stack, as opposed to the more traditional vertical design in normal MAC protocols. The horizontal design allowed a lot of control at application-level, with the trade-off that an application was able to tweak core parts of the MAC layer that could potentially introduce significant instabilities in the MAC, unless the application was fully aware of how the particular MAC would react to those changes. In the λMAC design, applications have large quantities of control - they can allocate arbitrary blocks of time and do pretty much whatever they like during this time - but in a way that preserves the integrity of the λMAC layer, as it is able to delay AllocateTime requests until it is a "good" (for values of "good" defined by the individual λMAC layer) time for the application to have control. The λMAC separation of control, with most timing control out of the hands of the application designer, allows for cleaner, safer, and simpler design.

Ee et. al [31] attempted similar goals, but for routing protocols. Their approach looked at providing a generic toolkit for building routing protocols, and for creating modules that could be used to piece together protocols, including the possibility of new hybrid protocols built from parts of earlier protocols. Their wish to do this as opposed to a framework design such as we proposed is possibly indicative of a wider variety of options in routing protocol design, as opposed to the relatively small set (time management) that we have identified here for MAC protocols.

Cross-layer design, interlacing MAC design with other protocol layers (typically routing [75, 95], although localisation protocols have also been integrated with MACs [5]),

is another direction that has seen some effort in MAC design. For some cases e.g. the ExOR transmission module (Section 3.3.2), the λMAC framework allows building modules within the MAC stack that are designed to work closely with other layers (the Guesswork routing protocol in Chapter 3 in this case), thus enabling limited aspects of cross-layer design without the tight bindings between layers (and subsequent restrictions on choice of MAC protocol) that is typical of most cross-layer design.

2.14 Conclusions

We set out to redesign and rethink how MAC protocols are designed for WSNs, to create a new and improved design concept, and to modularise common functionality. We have managed to do this, providing new capabilities and a refocused take on the role of a MAC in the WSN network stack. The reduction of the role of a MAC protocol to its core feature of time management, by separating out the Network Time layer to provide application-wide time synchronisation, as well as the Transmission layer modules to allow for clean separation of the logic required for features like Unicast, has given a new look at an old topic.

From our testing here, we have managed to show that our initial attempt at a reference λMAC layer (λT-MAC) was able to achieve similar performance, both in terms of data rates and power usage, to a traditionally designed MAC protocol, but with a significant decrease in complexity. Lines of code is not always a good indicator of system complexity, but the reduction of duties required of λT-MAC v.s. monolithic T-MAC is. We were also able to show that LMAC, a TDMA-based protocol that we expected to be a difficult case, turned out to be not so hard to implement. Some modifications to our existing work were required, and more work with λLMAC is required, but it has already managed to show good performance v.s. existing work with traditionally designed implementations.

In contrast to the areas that we will look at in later chapters, previous MAC protocol design turned out to be less abstracted than the more optimal form we have shown here; as opposed to higher levels of the protocol stack which tend to have the problem that abstraction has been taken too far, and needs reducing to a more suitable level. We postulate that this has to do with the perception that a MAC protocol is a very low-level, heavily hardware-dependant protocol, that should be tightly integrated with the particular radio hardware. TinyOS 1.x, with its multiple network stacks for different hardware platforms is a clear expression of this belief. In contrast, our example implementation here is almost entirely platform-independant code, with only the most basic of timers and radio interfaces being platform-specific, and we still have similar performance to tightly integrated stacks.

By implementing two significantly different MAC protocols, we have shown that our framework is sufficiently generic to be used by the wider community as a general-purpose MAC creation framework. Especially for experimental platforms, the importance of allowing people to extend existing work without having to reinvent the wheel cannot be overemphasised. The emphasis on the use of platform-independant code is a key ele-

ment of this, as it makes porting to a new hardware platform (which happens relatively frequently in WSN research) not very difficult at all.

2.14.1 Future Work

Certain adaptions of the AllocateTime interface would allow further integration with other MAC protocols, and enable more efficient implementations of TDMA-based protocols. Extending the AllocateTime interface to provide more information about what nodes are the destinations of the packets to be sent during the interval would allow better allocation by TDMA schemes, and possibly noting that certain time slots are more reliably allocated than others, as most TDMA protocols have more reliable guarantees about the lack of other nodes transmitting v.s. CSMA protocols with carrier sense. In general, finding better ways to specify more information about the usage patterns for a given AllocateTime slot in a generic way to the λMAC layer will help smarter λMAC protocols allocate time more effectively. We would also like to explore possibilities for more types of Transmission modules.

We hope that one of the side effects of our creation of the λMAC framework will be the creation of more MAC protocol implementations for TinyOS, as many new MAC protocols are currently only implemented in simulation, and simulation is a poor guide to how something as low-level and radio hardware dependant as a MAC protocol will behave on real hardware.

Chapter 3

Routing

In this chapter: We deconstruct the idea of "unicast links" between nodes, build a new set of sending primitives, and use them to build an energy-efficient routing protocol.

Packet routing is a problem common to all networks; big and small; wired or wireless; electronic or physical - we all want to be able to transfer items from point A to point B. The wider field of distributed systems deals with a problem similar to the real world - how do I get from any point to any other point by passing messages from one node in the system to adjacent nodes. The canonical example here is the Internet, with clients wanting to get data from servers, and needing a way to establish a usable route between the two. For most wireless sensor network problems, the required routing is often less complicated given the data patterns, but we need more efficient solutions for the paths to be viable.

In all cases of routing, you want to do this efficiently, but the metrics for efficiency vary from situation to situation. For conventional networks, the emphasis is usually on bandwidth and minimum latency, along with generally considering all links between nodes as being perfect. For situations where the links are imperfect (e.g. wireless links), the imperfection is masked by retries and acknowledgements. In effect, a "perfect" link with reduced bandwidth and increased latency can be created from an imperfect link. TCP [101] is the most commonly used example of this, being used to create perfect end-to-end links from imperfect links despite its known issues with wireless systems [9, 10, 29, 82, 146].

For sensor networks, high bandwidth is not very important, low latency is often less important, and due to the low-cost hardware our radios are often of lower quality than most other wireless networks. As we discussed in Chapter 1, the most important metric for sensor networks is power consumption, which conflicts with any attempts to actually do anything. To create a minimum power system, we need to create a system with

Most of the content in this chapter has been published as "Guesswork: Robust Routing in an Uncertain World" by Tom Parker and Koen Langendoen at the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005) [2]

non-zero bandwidth and non-infinite latency, while doing the minimum effort over a zero-bandwidth/infinite-latency system such that we can provide what the application needs and (preferably) nothing more. The strictness of these metrics may vary, given the possibility of state space explosion when many different choices are possible for a route - the routes being chosen often only need to be "good enough", not perfect. If for example we only need a system to run for two months, reducing power consumption down to the level where it could run for a year is not necessary, especially if we can improve the level of services (bandwidth, latency) that we provide to the application.

3.1 Sensor Network routing

Given the new emphasis on power management as the key metric, most routing protocols have a number of significant problems when applied to Wireless Sensor Networks. The major trade-off for non-data overhead in a routing protocol is between adaptability to changing network conditions and maintenance overhead, but this trade-off illustrates clearly the issues with WSNs vs. conventional wired networks (limited power, limited memory storage, limited processing, etc). For conventional networks (e.g. the Internet), adaption is minimal, as most of the nodes tend to be statically located and maintenance overhead can contain quite a lot of data without significantly impacting the ability of the network to transmit the data required by the application.

Given the scarcity of power availability to WSNs, adapting to changing network conditions is difficult, as this often requires the re-dissemination of routing information for a sink or other critical node across a significant proportion of a network. The overhead required to keep standard routing information up-to-date when the actual amount of data being transferred across said network is taken into consideration is often unacceptable. This optimising for very low traffic rates, as opposed to traditional networking, which tends to optimise for high traffic, changes the balance between what was previously considered as acceptable overhead for "normal" data rates, as well as the fact that any overhead reduces the lifetime of the network (due to finite power limits). This all means that normal approaches need to be revisited.

We already showed earlier a little bit of how the approaches for WSNs are different (Section 1.2 on page 4). Sensor networks use two patterns for routing their data: local neighbourhood co-operation and source-to-sink. The former is generally trivial, or can be implemented trivially for k-hop neighbourhoods (k is often, but not always 1 in this case), but the latter is more difficult. Source-to-sink routing is used to implement one of the core design goals for most sensor networks - getting data about the environment around the nodes to somewhere where analysis can be done on it. Sometimes some level of analysis is done within the network (which we cover in Chapter 6), but often we need to move a significant proportion of the gathered data to a sink node that is generally connected to the wider world.

A significant number of protocols have been suggested for sensor networks, some of which are developed from existing network routing protocols, and others that are de-

signed towards the specific problems of sensor networks [3, 4]. The protocols can be classified into four groups: basic, hierarchical, data-based and geographic.

3.1.1 Basic

The protocols in the "basic" group are mainly those ported from earlier work in mobile ad-hoc networks and related network applications. Examples of this group include Flooding and Gossip-based protocols [44]; AODV [96] (Ad-hoc On Demand Distance Vector); DSDV [97] (Destination-Sequenced Distance Vector); and DSR [56] (Dynamic Source Routing).

Many of the basic protocols are relatively heavyweight, and a common method for route discovery is flooding a local area to do a greedy search for a particular destination node. Also, some (e.g. DSR) will require a complete list of nodes to route through to be stored in the packet that is being transmitted. This group of protocols were originally designed for the any-to-any routing case, and although these solutions will work with source-to-sink routing, they are substantially inefficient. However, they are simple, and in the difficult environment of sensor networks, a simple yet robust approach with multiple retries of failed packets can sometimes be a viable option, despite the energy requirements.

3.1.2 Hierarchical

Hierarchical routing protocols for sensor networks mainly consist of variations on the theme of clustering techniques [38, 43, 149]. Subgroups of nodes within the network talk to elected "cluster head" nodes, who then forward messages through a backbone network built up by routing through cluster head nodes. The major advantage of this technique is that most nodes only need a 1-node routing table (the local cluster head), and the cluster heads only need to talk to a smaller list of nodes. By reducing the list of nodes that need to be communicated with, a large network can exhibit the efficiencies of a network an order of magnitude smaller. Additionally, nodes within a cluster may well be able to sleep for larger periods of time due to the lack of need to forward messages for other nodes. Some clustering techniques take this one step further, and create clusters of clusters - effectively repeating the same process, but using clusters rather than individual nodes as the smallest units [12].

Hierarchical techniques have several disadvantages, including the additional overhead for cluster head elections, as well as the need to change cluster head every so often (or the cluster head nodes will run out of power far earlier than the other nodes). Clustering is a relatively expensive method for routing in terms of setup costs, but theoretically these costs will save energy over the long term given sufficient message traffic. Some protocols have been developed to reduce the overhead, including Passive Clustering [38] techniques to piggyback election data onto other packets, but the underlying trade-off between cost of setup v.s. energy saved over time remains.

3.1.3 Geographic

Geographic routing protocols use the physical locations of nodes to determine routing paths. They therefore require nodes to have knowledge of their locations - however, as this is a useful piece of information to other levels of the application stack, and with the possibility of local co-ordinate systems as well, this is not so much of a problem. Some [87] use the idea of a trajectory between the current location and the destination node to calculate an ideal vector to travel along, which represents the shortest path in space between the two points. The next hop neighbour is chosen using closeness to the ideal trajectory, as well as distance from the destination node to choose the best candidate node. Others [59] simply use distance to the destination node.

One of the major problems with geographic routing is that nodes are not evenly spread out, and it can often be easy to reach a node that was a local optimum for reaching the destination node, but has no neighbour nodes that are closer to the destination, due to an "empty" region in the topology of the network. The techniques used in this situation contain the major differences between different geographic routing protocols. GPSR (Greedy Perimeter Stateless Routing) [59] then routes around the perimeter of the empty region; Trajectory Based Forwarding [87] uses a combination of greedy forwarding and limited flooding to work around the problem.

3.1.4 Data-based

In contrast to the other groups of protocols addressed so far, data-based protocols attempt to make routing decisions not based on the sink needing to receive all of the data, but on the idea that the sink node is only interested in certain subsets of the available data. These protocols commonly feature advertising messages by nodes with metadata describing what information they have available, or requests from the sink for certain types of information, including concepts like the idea of the sink being "interested" in certain forms of data.

SPIN [66] was one of the early works in this area, using an advertising mechanism to locally disseminate information about a node's data. Information from a node however only reached a limited distance, and when the sink was far from the source node then the information did not always arrive. Directed Diffusion [54] improved on the basic idea by making the sink initially propagate interest information across the whole network, and using this interest flood to determine the complete route that a particular piece of information should take to an interested node. The lack of fixed routes, and the use of fully data-centric routing was a significant milestone in sensor network routing.

3.2 Problems

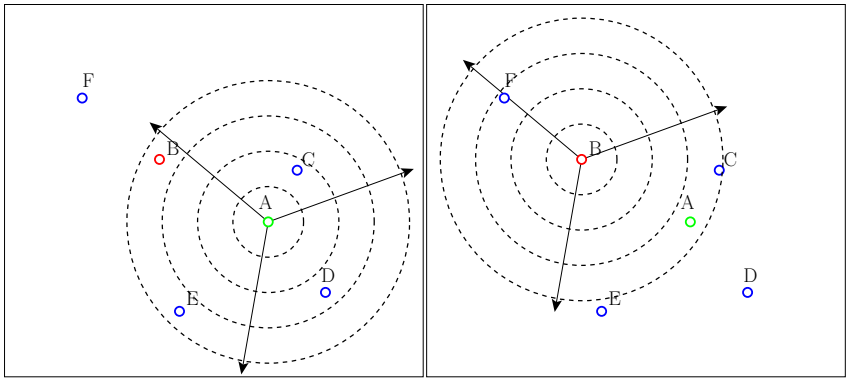
All of the protocols listed above - with the exception of Flooding-based protocols - have one core problem, in that their basic mechanism for transferring data from one node to another is a unicast transmission. Unicast, in a wireless network, is an abstraction

built on top of a series of broadcast transmissions. If one of these transmissions fails (a relatively likely event) then the entire sequence fails, and is generally just restarted. If we have uni-directional links (as opposed to the bi-directional links assumed by most wired protocols), then this will keep on happening.

The concept of a "link" between two nodes is itself another abstraction. The concept of a "wire through the air" or the frequently quoted "no cat" conception of radio[1] is an idea with no actual basis in the physical world. The actual sequence of events for communication between two nodes, A and B, is closer to the following (shown in Figure 3.1):

1. Node A broadcasts a message which contains Node B's id. Nodes B,C,D and E receive the message.

2. Node B broadcasts another message, which contains Node A's id. Nodes A,C,E and F receive the message



(a) Step 1 (b) Step 2

Figure 3.1: Message Sequence

Several points are of note here. Firstly, Nodes C, D, E and F get messages that they will then discard as they are not destined for that node. Messages not for a node look exactly the same as messages for a node (from the point of view of the radio hardware) but some of their encoded data indicates a different node id. Secondly, if either step fails i.e. the node that we wanted to get a message to in each case fails to receive the message, then the whole sequence is repeated, and may well fail again. However, one of the other nodes may have received a message that was not destined for them, and if they were smart enough to realise what has happened and use that message as opposed to discarding it,

[1] Albert Einstein, when asked to describe radio, replied: "You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat."

then they could be the node repeating this sequence of messages, as they may well have better odds of succeeding with the transmission.

Flooding protocols, on the other hand, have broadcast transmission as their basic mechanism for transferring data. This is, in a way, a better idea because unreliable broadcast (being one step down the "abstraction stack") is an intrinsically more efficient operation than unicast. However, flooding is not a smart protocol, with the repeated transmission of messages containing no feedback mechanism to either a) determine if a message has already reached the destination or b) succeeded at the end of a sequence of repeated transmissions, both removes the improved efficiency and does not provide any guarantees (or knowledge) about transmission success/failure.

The abstract concept of a link is also connected to the abstraction of a data packet as a physical object which we discussed earlier in Section 1.3.2. As we showed above, wireless transmissions are based on broadcast transmissions, which is incompatible with the notion of a single physical object, and a better mental model may be relating wireless transmissions to being like a person speaking, and that their voice may be heard by other people who are nearby, but this also has flaws (as shown in Section 1.3.2).

3.3 Partial Solutions

The problem of the lack of proper "links" has been previously examined, and some partial solutions towards solving these issues have been already proposed. In this section, we will look at two of those, and show the remaining issues.

3.3.1 ETX

Most routing metrics have cut-off levels - a link is considered to be arbitrarily good or bad. For most realistic scenarios, this is often not the case [26]. Sometimes we will have a lot of good links, and then we can discard more, sometimes we will have a very bad connection to the sink node, but we still need to be able to communicate.

Expected Transmission (ETX) count [25] provides an improved metric for routing decisions, based on the expected number of transmissions via a particular next-hop node to reach a particular destination node. This allows for adapting to the complete variety of node link conditions [155] - everything from perfect links to dealing with broken and partial links.

A partial link for example will increase the ETX value for a path because it is not always totally reliable, and broken links can be handled by significantly increasing the ETX value of a node that cannot find its next-hop neighbour, as a node with no next-hop neighbour can thought of as a node with a with a very high ETX value to the destination node.

For example, we may also have cases like Figure 3.2, where a shortest-hop algorithm would pick Route 2. But, this has a higher ETX and therefore a higher average transmission count than Route 1, despite the fact that it requires fewer hops.

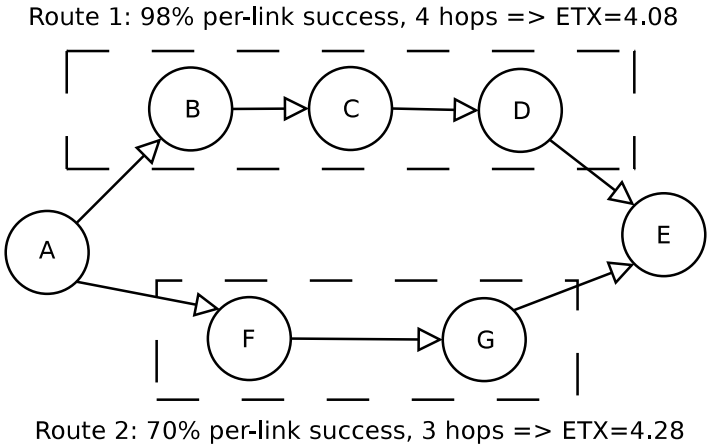


Figure 3.2: Not all links are equal

ETX also allows for dealing with heterogeneous networks with gateway nodes (nodes with a faster link over another network to the sink, and usually an external source of power) - these can simply declare their ETX cost to be very small, as their faster link and external power means they represent a much better route to the sink.

In the original ETX specification [25], De Couto et al. used probe packets to determine what the ETX cost for a link would be. This is expensive, with an unacceptably high overhead cost for low transmission rate scenarios i.e. most WSN applications. Additionally, because the probe packets are generally a lot smaller than actual data packets, probe packets do not necessarily provide an accurate picture of how good a link is for data packets. Other options include monitoring the data packets being used (which has the problem that initial packets will have an inaccurate ETX value) or approximations based on radio signal strength values of packets received from the destination node (which assumes the existence of a valid mapping between signal strength and reception probability).

3.3.2 ExOR

ExOR [13] uses a “one send, many replies” idea to do localised greedy routing, providing a better utilisation of the basic broadcast medium available to WSNs. This is based on the idea of a set of neighbours receiving a message from a sender node, all of the specified neighbours sending an ACK for the message, and the best next-hop node for a particular destination (of the set that receive the message) gets chosen, without having to do additional communication beyond the Data/ACK sequence already performed.

Figure 3.3 shows an example timeline for an ExOR packet. The DATA segment is sent by the sender node, which includes a list of neighbours in the order of how many hops it would take to get from that neighbour to the destination node. The neighbours (nodes 3,2,4 and 1 in this case) reply in the order that was specified in the DATA message.

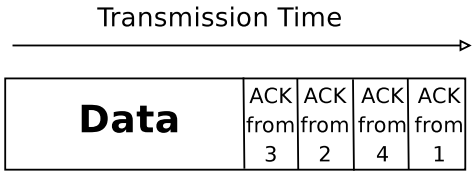


Figure 3.3: Example ExOR packet timeline

If a neighbour node does not hear any reply messages for nodes earlier in the order than itself, then it forwards on the message. If all of the neighbours can hear each other, then this will result in only one sending on the message. To help this process, if a node has already heard a reply message when it sends its reply message, then it replies with not its own id, but the id of the best (i.e. earliest in the neighbour list) node that it has heard a reply message from.

The use of shortest-number of hops by the originally proposed version of ExOR (which is not necessarily a good choice for routing decisions, as we discussed on page 54) relied on the knowledge of a local node about its neighbours, because it required that the neighbour list of an ExOR packet should always be in order of preference, based on information available at the start of the ExOR sequence. This creates a situation which is prone to allowing out-of-date information to be maintained for longer than it should be and also stops a number of possible useful extensions to the protocol, as this relies on locally-cached data about neighbours v.s. querying the neighbours for their current state.

3.4 Generalised ExOR

In this section, we present a series of alterations to basic ExOR, broadening its scope and allowing for a variety of "choice functions", including one that uses ETX values. We then use the generalised form to create new useful primitives for routing protocols - specifically, a reliable broadcast mechanism, and a reliable way to do source-to-sink data transmission.

3.4.1 Choice Functions

Generalised ExOR specifies the neighbour list in an arbitrary order, and the neighbours respond with a particular value (˜1 byte of data for most choice functions). Which value, and the resulting actions depending on that value, depend on the particular effect that is required. A choice function defines how the protocol responds to ExOR messages, including how a particular given node receiving an ExOR message will then decide whether the message should be forwarded onto other nodes.

Some useful possibilities include:

- Sending the lowest hop count that this node has heard (or its own if it has heard none so far), and forwarding if we have the lowest hop-count heard. This is the original ExOR choice function.
- Sending the lowest ETX value that this node has heard (or its own if it has heard none so far), and forwarding if we have the lowest ETX value heard. We refer to this as ExOR-ETX.
- Sending a bit-field representing the set of nodes that this node has heard an ACK for (including its own), expressed as a series of bits in the same order as the original transmitted set of neighbours. The original sender can then OR together the received bit-fields and infer the list of nodes that have received the message, thus allowing for a reliable broadcast mechanism with reduced cost compared to unicasting to every neighbour. The OR'ing together of the received bit-fields allows the sender to get a good picture of what nodes have received the message, even in the case of asymmetric links. We refer to this as ExOR-Bcast.

3.4.1.1 Multi-hop Reliable Broadcast

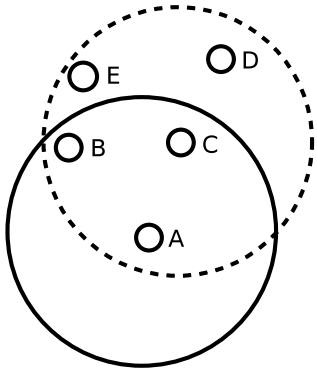


Figure 3.4: Multi-hop broadcast example

ExOR-Bcast can further be improved as a method for reliable broadcast over multiple-hops i.e. a message flooding scenario, by overhearing of broadcast messages from neighbours. This can be used to reduce the set of neighbours that we need to send a message to by eliminating those that we have heard an ACK for (or have seen another ACK that contains the relevant bit set for that neighbour).

For example, see Figure 3.4. *A* is a node that has sent an initial ExOR-Bcast to *B* and *C*. *C* then proceeds to do ExOR-Bcast to *D* and *E*. If *B* overhears any of the replies from *D* or *E*, it can determine if they have heard the broadcast, and so therefore it may not be

necessary for *B* to do ExOR-Bcast at all, or it can at least reduce the set of neighbours that it needs to send the message to.

3.4.1.2 ExOR-ETX

With ExOR-ETX, if two nodes have the same ETX value (which is a quite likely scenario, especially in cases with reliable links) and both receive an ExOR-ETX message, they will both by default decide to forward on the message, resulting in duplication. This may occur many times, resulting in an significant increase in the number of messages sent in the attempt to successfully deliver one message. In a number of likely scenarios this is not so much of a problem, as in many cases with equally good next-hop nodes they will both be travelling a similar path to the sink, and so the two routes will likely overlap later on and the duplicate message can be filtered out at that point.

However, a better solution to the duplication problem is to change the metric field for ExOR-ETX, reserving one bit (usually the high bit) for use as a "sender" bit. A node sets this bit to indicate that it has taken responsibility for making sure that the message is sent, not that it will necessarily send the message. If a node has decided it is to be a sender, and it sees no other replies with a sender bit, then it sends out the message. If however a node sees another message with a sender bit and an equal ETX, then it must make a decision whether it is a sender or not. This decision is based on an arbitrary, but fixed (for a particular network) function for any given network, that will always be able to decide a single sender node. The current implementation uses node ids, and the node with the highest id is the sender.

3.4.2 Inverted ExOR

One problem with the ExOR methodology is what we call the "neighbour bootstrap" problem. When a network has very little traffic, then it may be the case that a node does not receive any messages from its neighbours, or only from a subset of them, and that the node needs to be able to send messages to the neighbours that are being quiet and not announcing their presence. Communicating with these nodes is difficult due to the fact that ExOR requires neighbour knowledge for the scheduling of the ACK messages.

To combat this problem, we provide another variant on ExOR using an inverted neighbour list i.e. the neighbours that are in the list in the ExOR message should not respond. For a bootstrap scenario with no currently known neighbours, the list may well be empty. The message specifies instead a time period (specified as the number of control packet intervals) after the message has been sent, in which nodes not in the list should respond. Each node that responds picks a random interval within the specified time period to respond (using slotted aloha [112] with the transmission time of a control packet as the interval).

With this random choice, the likelihood of collisions increases significantly, but the focus of this method is acquiring information from some neighbours, in order to reduce the problem of having sparse neighbour information. In the case that we have not received information from enough neighbours, then the message can be repeated, adding

the successful neighbours from earlier stages to the "do not send" list each time. The reasoning behind this is to reduce the number of collisions, by telling the neighbours that we have already received data from not to reply.

An additional constraint on the design of the control function that can reduce the collision rate is only having neighbours that have something useful to say responding (as opposed to the "everyone should respond" models used for standard ExOR) e.g. in a message routing scenario, only nodes that have a good route to the sink should respond.

3.5 Guesswork

Now that we have these improved primitives, we can build a better routing protocol. Guesswork is an adaptive, probabilistic routing algorithm for wireless ad-hoc sensor networks, using local knowledge of best guess next-hop nodes to efficiently implement source-to-sink routing. Guesswork is based upon ExOR-ETX (Section 3.4.1.2), and our multi-hop reliable broadcast (Section 3.4.1.1) mechanism, and extends these to allow for adaptation to changing link qualities. Guesswork also uses a technique for creating and adapting ETX values for destination sinks over time using information from the data sent over a network, and so be able to adapt to changing network scenarios. It is designed to work efficiently in a wide variety of application scenarios, being able to cope with low quality links as well as both static and mobile networks, and all with a minimum amount of overhead.

We firstly introduce the main framework of the Guesswork protocol, show how this works with the λMAC framework (Chapter 2), and then present simulation results for Guesswork and other routing protocols on top of a variety of MAC protocols.

3.5.1 Initialisation

In order to formulate a good solution to the source to sink routing problem, first we need to look at our discovery process for the sink nodes themselves. One way is that all nodes are automatically told about all of the sink nodes at startup i.e. by hard-coding the sink node information into the nodes. This is however inflexible, assumes that we already have the sink information at the time of system deployment, and is generally unsuited for ad-hoc WSN scenarios.

A better solution is to do initial flooding of the sink information to the network. If we have a reliable broadcast mechanism then we can reduce the flooding to a single instance per sink. When a new node starts up, it can request the sink information known to its neighbours. A possible extension to this is the inclusion of query information within the sink broadcast (i.e. what kind of data a particular sink is interested in, similarly to Directed Diffusion [54]), but for now we are only considering the situation where all sinks want all the information. With this initial sink to sources flood, we can discover the initialisation ETX values (one per source node) for a sink.

This information may not be perfectly accurate (asymmetric links will cause problems for example), but it represents a reasonable first approximation to the correct current ETX for a sink.

3.5.2 Message Transmission

Guesswork relies on the existence of a reliable protocol to implement source-to-sink transmission once an ETX value for a particular sink is known by a source node. Packets being transmitted with Guesswork also contain a TX count so far for the packet.

3.5.3 Adaption

Nodes have an initial value for the ETX value for the sinks in the network (from the Initialisation phase), but this will change over time as the network alters (node failures, broken links, new links, etc), so we need to be able to adapt the ETX value for a sink over time. One way this can be done is by propagating route update packets back to source nodes when a successful transmission to a sink node has differing values for the number of transmissions used v.s. the original ETX value. The route update contains the actual TX count that was used for the particular source to sink route.

Route discovery for the sink to source route can be done by the nodes on the path (P) that a particular source node (A) uses to get to a sink node (S), because all of the P nodes will have a TX count for the incoming packet from A, and so they can record this as an estimated ETX to return to A. The new ETX to S for each of the nodes along the path can also be updated, by subtracting the ETX value that the packet had on the way in at a particular node from the total TX value that is being reported in the route update packet.

If we have a route update, then we have a new ETX value. This can now be used to update our current recorded ETX value, using a learning function based on these two values. This is done because a single changed route cost does not necessarily mean all transmissions to that sink will be equally low/high. The simplest form of this is:

$$ETX_{updated} = ETX_{new} * LearningConstant + ETX_{old} * (1 - LearningConstant)$$

where the value for *LearningConstant* is ≤ 1. Earlier work in similar areas [103] suggests values in the 0.2 to 0.4 region, but more experimental testing would be needed to determine suitable values for typical applications.

An additional optimisation is the aggregation of route update packets, as these are only used to propagate the ETX value back along the sending path. Instead of sending one route-update for every message with a changed ETX, an aggregate update consisting of an averaged ETX (with a packet count) for a set of packets can be sent back to the sender node. A suitable mechanism for this would be waiting until no packets have been received from a sender for x seconds (10 for example) before sending the aggregate route update. Also note that if the aggregate ETX is very close (e.g. $<5\%$ change) to the original source node's ETX for our sink, then we can simply discard the route update completely as no update is necessary. These measures reduce the number of route-update

packets so that they are only generally sent when the network is changing and during periods of network stability they do not need to be sent at all.

3.5.4 Failure Resilience

New route discovery in the event of a failure (no responding neighbouring node has a lower ETX than the current node) consists of bouncing the packet back up to the previous node along the chain and repeating the send sequence. In this case, we can also update the failed node's ETX value for the destination sink by using a value for ETX_{new} that is higher than any other ETX value that we have already seen. Possible values include $ETX_{Highest} + 1$ or $ETX_{Highest} * 2$, but again, experimental testing will also be needed to resolve better values for this. We should now have an increased ETX (due to the updating from the failure) and so another possible neighbour will probably be chosen instead.

For example, if a node *A* has a message for sink *S* and it chooses node *B* as its next-hop neighbour, but *B* is unable to forward the packet, then *B* updates its ETX for *S* to a much higher value, and bounces the message back to *A*. At this point, *A* will go through the message-forwarding mechanism again, and *B* could potentially be chosen again, but given that *B* has just had its ETX for *S* significantly increased, it is likely that another node will be chosen.

3.6 Implementation

So far, we have specified what we want to do, and some high-level details of how we will implement these choices, but another important factor to consider is that in most WSN systems there are existing other protocols that we must interact with.

Most existing WSN routing protocols [16, 54, 59] come in two forms: weak and strong binding to the MAC layer - those that just treat the MAC as a black box that will send packets, and those that rely heavily on one particular MAC. The former methodology is unsuitable for ExOR use as the uncertain delay between one packet transmission and the next (due to MACs doing things like sleeping [24, 148] and letting other possible parts of an application use the radio), and the latter would reduce the flexibility of ExOR to interact with a variety of protocol stacks (which is a problem given the heterogeneity of current WSN systems).

We will therefore use the λMAC extensions from Chapter 2, in order to provide a suitable level of control, while still allowing compatibility with many different MAC protocols.

3.6.1 Building ExOR

Using the λMAC extensions, we can now build generalised ExOR as follows. In all cases, we assume that at startup we call sendTime() with the payload length of a reply packet (1 byte for most choice functions) to get the value *ReplyTime*.

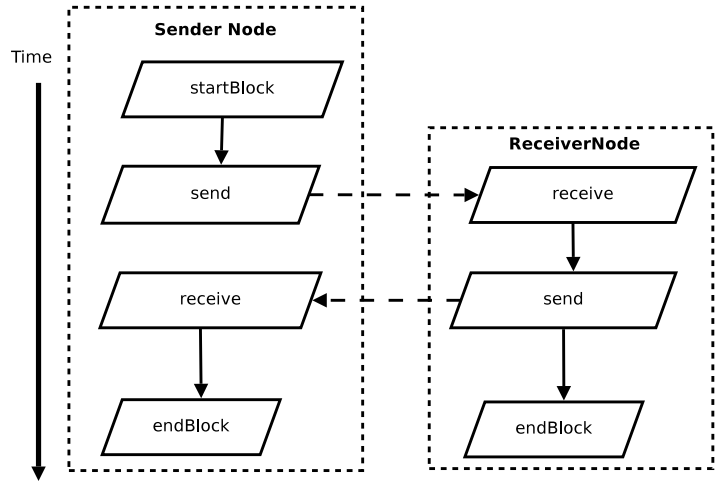


Figure 3.5: ExOR time line example

ExOR Sending node:

1. Call requestBlock() with Time equal to $sendTime(packet\ length) + ReplyTime * neighbours$ (where *neighbours* is equal to the number of neighbours listed in the ExOR packet - see Section 3.3.2)
2. On startBlock, call send() with the packet that we want to send.
3. On endBlock(), perform whatever cleanup operations are associated with the used choice function (e.g. for ExOR-Bcast, record which neighbours we now have additionally managed to send the message to).

ExOR Receiving node:

1. On a receive() event, where the incoming packet is a message from a Sending node, then record the sender node id
   a) If invert is switched off for the message, and this node is in the neighbour list, then set z to be our index in the neighbour list.
   b) If invert is switched on for the message, and this node is not in the neighbour list, pick a random value between 0 and n as z, where n is the maximum number of slots for the reply packet.
   c) If neither a) nor b) apply, return 0 (go to sleep until endBlock).
2. If $z > 0$ then
   set a 'reply packet' timer for $z * ReplyTime$ ms
   else
   send a reply to the sender with send()
   return $n * ReplyTime$ (wait for entire packet sequence time)

3. If a 'reply packet' timer goes off, then send() a reply to the recorded sender node.

For all types of node, on a receive() event, where the incoming packet is from a Receiving node, then apply the control function as appropriate for the ExOR variant in question (see Section 3.4.1), and return 0 (go to sleep until endBlock).

3.6.2 Building Guesswork

Guesswork (as we discussed in Section 3.5) requires a reliable broadcast mechanism (ExOR-Bcast), and a method for reliably transmitting messages using ETX (ExOR-ETX). We have shown how to build these, but there are a few remaining details of the Guesswork use of these algorithms to be mentioned.

Namely, the choice of how many neighbours to have in a neighbour list. For ExOR-Bcast, the answer is generally fixed - in our implementation, we stick to a maximum of 5 neighbours - more would make information spread faster, but as longer messages tend to have a greater probability of failure (due to interference from other nodes, and effects like the hidden-terminal problem (Section 2.1)) this appears to be a good value. Also, because we actually want to talk to everyone in ExOR-Bcast, talking to too many neighbours is less of an issue. Additionally, as ExOR-Bcast is generally only used by Guesswork during application setup, optimisations to this will have a minimal effect.

However, the choice of neighbours used by ExOR-ETX is more important - partly because this variant is more often used, and partly because ExOR-ETX only actually wants to talk to one neighbour and the others are only for redundancy. As excessive redundancy is overhead, a good value for this is application specific, but as we would like the algorithm to work with a variety of applications, a method for automatically deciding on this value is useful. One option is using the following algorithm:

1. Each node starts with a neighbour count value (e.g. 5), which is used to determine how many neighbours are used in ExOR-ETX
2. A node keeps track of the last known ETX value for each of its neighbours, and every time an ExOR-ETX sequence is executed, it checks which node its cached values for the neighbour ETXs would have chosen v.s. the actual winning node.
3. Every time the node guesses correctly, it decrements the neighbour count value (down to a minimum of 2) and every time it gets it incorrect, it increments the neighbour count (possibly up to a maximum value e.g. 10)

The idea behind this is that if a node can correctly guess the correct node to send to next, then the network is probably moving towards a stable configuration with stable links. If a node guesses incorrectly, then it is probably worth expanding the neighbour list to check against other nodes. The neighbour list in ExOR-ETX effectively represents a "candidate node" and a list of backup options. Therefore, giving more backup options in a unstable situation, and less in a stable scenario is a good idea. The limit of 2 neighbours as a minimum ensures that there is always a backup neighbour, and avoids the node collapsing towards the fixed route scenarios that Guesswork intends to avoid.

For the purposes of the simulation testing, we created a fixed-length neighbour list and did not implement this extension.

3.7 Results

We proceeded to test Guesswork against other routing algorithms, in combination with a series of different MAC protocols. We wished to test Guesswork against a TinyOS [49] implementation (similarly to our MAC work in Chapter 2), but a dearth of comparable routing protocols for TinyOS, as well as some remaining stability issues in our implementation of Guesswork for TinyOS limited what we could do.

Instead, we decided to stick to simulation. This also allowed for testing with more nodes, as our testbed is currently limited to only 24 nodes. The simulation framework is based upon Positif [72], but extended and altered to work with routing protocols rather than localisation protocols. The MAC protocol implementation is taken from earlier work on MAC protocols [24], which has been extended to interface with Positif to create a unified simulation framework.

	Parameter	
Protocol	Name	Value
AODV	"Hello" messages	Disabled
Gossip	Fanout	2 for 1st 5 hops 1 afterwards
	TTL	20 hops
Guesswork	Neighbour list size	5 (fixed, no adaption)
S-MAC	Frame length	1000ms
	Timeout	100ms
T-MAC	Frame length	610ms
	Timeout	15ms

Table 3.1: Protocol parameters

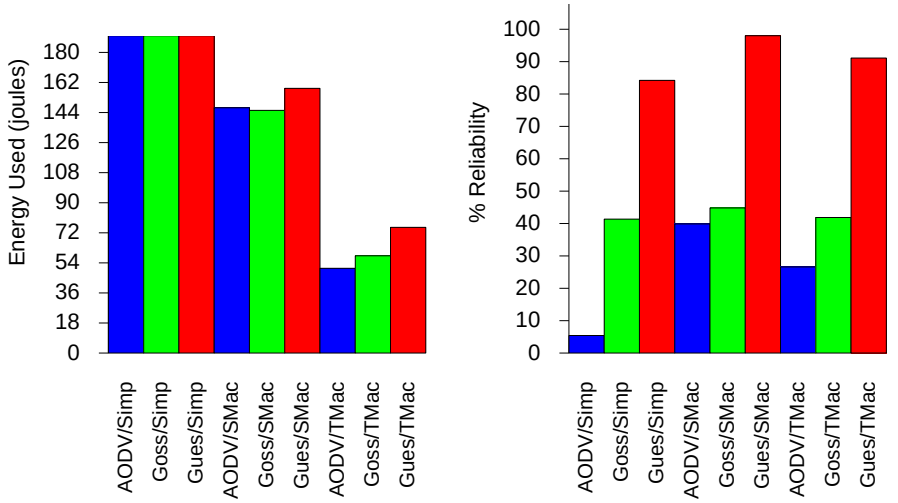
In each case, we have a simple routing test, consisting of a source transmitting a packet every 10 seconds, until it has sent a total of 20 packets, and sending towards a single sink. We also considered a number of other possible scenarios, but decided to stick with only the simple scenario as it was the most applicable to many sensor network applications (v.s. more complicated scenarios which would have posed significant difficulties choosing a widely representative scenario).

We tested 3 different MAC protocols S-MAC [147], T-MAC [24], and a "simple, no carrier-sense" MAC (to provide a baseline comparison). The other routing algorithms being compared against are AODV [98] and Gossip (random walking with limited fanout). The AODV implementation was ported from the existing implementation for the GloMoSim [151] simulator. Given their lack of a sink-discovery mechanism, both AODV and Gossip were informed of the address of the sink at the start of each test.

56 nodes are present in each test, in a 50x50 area, with maximum radio range set at 14. The per-link reliability is set at 80% for all tests i.e. a random 20% of all packets sent by the nodes are discarded, in order to simulate imperfect links. Note that 80% per-link reliability is a level that would be considered "good" by most algorithms that classify links as good/bad.

Each simulation is run for 300s before termination, and each result is the average of 20 runs of the particular combination of routing protocol and MAC protocol. The other parameters for the protocols are given in Table 3.1, but a couple are worth discussing further - AODV has its "Hello" messages disabled, because they resulted in far too much overhead for our simple WSN example, and Gossip reduces its fanout to 1 after 5 hops as otherwise it ends up reducing to simple flooding and having far too large overheads.

We looked at two different evaluation metrics for the results: reliability (how many of the source messages get to the sink) and cost (power used for transmission and reception over all of the nodes in the experiment, using the listed costs for a typical node transceiver [110]). The produced graphs for these metrics have been altered in a few small ways for improved readability - some of the protocol names have been shortened ("Gossip" has become "Goss", "Guesswork" became "Guess" and the Simple MAC is described as "Simp") and the "Simple" power measurements have been clipped (due to its lack of power management, the Simple MAC uses approximately an order of magnitude more power than the other protocols, and so displaying it fully would reduce the amount of usable information on the other protocols).



(a) Energy Costs (b) Reliablity

Figure 3.6: Simulation Results

A number of interesting results immediately appear from the graphs. Firstly, looking at Figure 3.6a, we can see that the overall cost of the messages is dominated by the choice

of MAC protocol - comparing differing routing protocols with the same MAC protocol shows a slightly higher cost for Guesswork and a slightly lower cost for AODV, but the difference is not significant. These results are consistent with current theory regarding such factors as idle listening (as noted in [48] and Section 2.1), and show that despite the additional overhead of the multiple ACKs in Guesswork, it still represents a viable alternative.

The advantage of Guesswork becomes immediately apparent when we turn to Figure 3.6b. Given that AODV depends on the reliability of a link at route discovery time, it performs badly when faced with links that are "reasonable", and may not always succeed. 80% is a high enough per-link reliability to provide links that can be used for routing, but low enough to cause enough failures to make otherwise viable connections be often discarded. Using the "Simple" MAC (with its lack of carrier sense, and so hence much greater packet drop rate) reduces AODV to unusable levels. Gossip, with its simpler methodology, is able to get some results (the redundancy from fanout is a significant factor there), but only with a reliability of 40-45%. Guesswork, on the other hand, is able to react to even low reliability levels and work around these problems, with end-to-end reliability at 85%+, climbing to 95%+ with the carrier-sense capable MACs. We have performed additional testing with Guesswork at lower reliability levels, and were able to maintain successful source-to-sink routing with link reliability levels down to ˜30%, without altering Guesswork in any way.

To further the goal of integration with existing protocol stacks, and to facilitate additional testing, Guesswork is currently being implemented for TinyOS. The implementation is still being worked on (as we mentioned at the beginning of this section), but preliminary results from using the TNOde platform indicate a RAM usage in the order of 672 bytes (16.4% of total), and ROM usage of ˜10Kbytes (7.8% of total). This still needs further optimisation work, but represents reasonable early work.

3.8 Conclusions

We set out to try and design a routing protocol that would work in an energy-efficient way to provide reliable end-to-end routing without having true links between nodes. As links were previously an implicit requirement of most routing protocols, and they were being provided in an energy inefficient way for sensor networks as they were a flawed abstraction of the underlying systems, a routing protocol that did not require true links would be much better designed (from an abstraction point of view), but would require rethinking on how to route data.

We therefore built Generalised ExOR, building an improved abstraction for working with hop-by-hop connections without having any explicit requirements for links. To further achieve these goals, we also built the Guesswork algorithm, built upon modified ExOR, along with the use of ETX to implicitly acknowledge that reception rates for pairs of nodes are not a simple yes/no question but a probability value. Despite its reliance on a packet sequence not normally supported by MAC protocols (ExOR), we showed how to integrate Guesswork with a range of MAC protocols by utilising the λMAC framework.

We have created here a routing algorithm that performs reliable routing over significantly unreliable links, and without significant additional energy costs v.s. existing unreliable routing protocols, by using a routing primitive (ExOR) with improved abstraction for WSN platforms.

3.8.1 Future Work

We would also like to explore ways of implementing Guesswork on top of TDMA protocols (which have very limited support at this point due to the requirement for being able to allocate blocks of time during which a single node can both send and receive packets). More work also needs to be done with testing against both fixed/reliable networks and mobile networks, as the current testing is against what is a fairly realistic middle ground for WSN applications, but testing how well Guesswork performs v.s. protocols designed for a particular niche in the search space of possible routing problems is of interest, as Guesswork is designed to work well (high reliability and low cost) in a variety of scenarios.

Another interesting consequence of unreliable broadcast connections between nodes and the removal of a "link" as a viable abstraction is that the concept of a "neighbour" of a node is a much more fuzzy concept. Previously, a pair of nodes would be considered neighbours if they had a communication link. However, now we have a wide range of nodes that could be considered "neighbours", depending on how often a given destination node will receive packets broadcast by a particular source node. A particular node's set of neighbours is no longer a classical set, but a Fuzzy Set [150], with membership criteria according to reception rates for each "neighbour" node. Fuzzy Logic has already been considered for use in sensor networks [41, 81], but the implications of fuzzy neighbour sets have not yet been fully explored.

Chapter 4

Localisation

In this chapter: We challenge the concept of distance estimation between nodes, define "probability maps" for distance estimates, and build a localisation protocol that can handle inaccurate ranging data using probability maps.

Many possible applications have now been thought of for Wireless Sensor Networks (WSNs), and a significant number of them rely on location information in order to perform their designated function. The main purpose of a WSN is information gathering, and gathered data is only useful if you know what it applies to. For example, the data "the temperature has gone up by 10 degrees" is not very useful, but the information "the temperature has gone up by 10 degrees in room 3C" is a lot more interesting. Location information gives us a context, which allows us to actually use our gathered data. For example, monitoring room temperature can be used to control when to switch air-conditioning systems on and off. When detailed location information is present it might even be possible to personalise working conditions within a shared office (i.e. individual settings per cubicle).

Location information is important in many domains, hence various approaches have been proposed, of which some were even constructed and deployed on a large scale (e.g. GPS). In other systems, GPS would be an option, but given the relative costs of GPS units (which are comparable on their own to the costs for a sensor node), the power requirements, and the difficulty of using GPS indoors [76], sensor networks need new solutions to the problem of localisation. Within the WSN community, specialised localisation algorithms have been developed that address the problems associated with the lack of infrastructure (i.e. external hardware existing in the the environment that can be used to aid localisation) and the limited resources leading to incomplete and inaccurate information. A survey of initial approaches is presented by Hightower and Borriello in [47], and Langendoen and Reijers [72] studied a variety of algorithms in more detail.

Most of the content in this chapter has been published as "Refined Statistic-based Localisation for Ad-hoc Sensor Networks" by Tom Parker and Koen Langendoen at the Globecom 2004 Wireless Ad Hoc and Sensor Networks Workshop [1]

With WSN localisation, some nodes may be referred to as "anchor" nodes, and others may not. The difference is that anchor nodes have a reliable source of location information, and non-anchor nodes do not. Many localisation techniques rely on anchors, but others do not. So called "anchor-free" localisation systems rely on the idea of building a local co-ordinate system based purely on the existing topology of the nodes, which provides the nodes with a location within the local system. In practise however, that location information would require further processing to integrate it with other co-ordinate systems (e.g. latitude/longitude). In this chapter, we are concentrating on anchor-based rather than anchor-free localisation techniques.

A major problem with localisation techniques (both anchored and anchor-free) is acquiring accurate range information between pairs of sensor nodes. This can be done in a variety of ways, ranging from simple techniques like Radio Signal Strength Indication (RSSI), time of flight data for various sensor types (e.g. ultrasound), to more complex ideas like time of flight difference (which measures the difference between two incoming signals travelling at different speeds). In each case, there is generally some error in the ranging information, which localisation algorithms must be aware of and be able to work with.

In this chapter we look at a number of the limitations with many of the existing proposed localisation techniques, show how they are unlikely to work well with a number of application scenarios, and present a refined approach. This approach uses a combination of a mobile anchor scenario for anchor information distribution, along with statistical techniques for performing localisation with inaccurate range data. Simulations with our improved approach have shown significant reductions (in the order of magnitude range) to the required processing for performing statistic-based localisation over previous attempts, as well as improving the generated location information in situations with non-total anchor information coverage, making possible a wider range of applications.

4.1 Existing localisation methods

In this section we will have a brief look at existing localisation algorithms, with an emphasis on their capabilities regarding the handling of inaccurate range information and their ability to handle non-uniform anchor distributions, which occur in many mobile anchor scenarios.

Langendoen and Reijers [72] studied three localisation algorithms that can handle a low number of anchors (Euclidean, Hop-Terrain, Multilateration), and identified a common three-phase structure. First, information about the anchors is flooded through the network to determine the (multi-hop) distances between anchors and nodes. Second, each node calculates its position using the known positions and estimated ranges of the anchors, for example, by performing a lateration procedure (as with GPS systems). Third, nodes refine their positions by exchanging their position estimates and using the one-hop inter-node ranges. After these three stages, a subset of the nodes have location information that is considered "good" (i.e. reliable).

Euclidean [86] uses basic geometric reasoning (triangles) to progress distance information from the anchors to the nodes in the network, and uses lateration to calculate the position estimates; no refinement is included in the algorithm. Euclidean's basic safety measure against inaccurate range information is to discard "impossible" triangles generated in phase 1. Unfortunately, this happens quite often, leaving many nodes in phase 2 without enough information to calculate their position (distances to at least 3 anchors are required). The end result is that Euclidean is only able to derive an accurate position for a small fraction of the nodes in the network.

Hop-Terrain [86, 115] avoids the range error problem to a large extent by using only topological information in phase 1. The distance to an anchor is determined by counting the number of hops to it, and multiplying that by an average-hop distance. Next, the node positions are estimated by means of a lateration procedure. In the refinement phase, Hop-Terrain switches to using the measured (inaccurate) ranges to neighbouring nodes. To avoid erroneous position estimates affecting neighbours too much, the refinement phase uses confidence values derived from the lateration procedure (dilution of precision and residue). Hop-terrain works well for a regular network topology in which nodes are evenly distributed. This however is not the case for the majority of mobile anchor scenarios, resulting in the algorithm becoming increasingly less accurate as the regularity assumption starts to break down.

Multilateration [116] proceeds by summing the distances along each multi-hop path in phase 1. To account for the accumulated inaccuracies it does not perform a lateration procedure, but instead uses each distance to specify a bounding box centred around the associated anchor, in which the node may be located. In phase 2, these bounding boxes are simply intersected and the position estimate is set to the centre of the intersection box, followed by a refinement procedure in phase 3. Multilateration's effectiveness with varying errors in range measurements will depend on the exact nature of the errors. If many of the measured distances are larger than the true distances, then Multilateration should be able to cope with the problem (as the true distance will still fall within the bounding box). However, in general, ranges are likely to be both under- and over-estimated (our current model treats both as equally likely), and Multilateration is less likely to be able to cope with under-estimation, and so will result in possible location information being discarded due to contradictions between ranges with varying errors.

4.2 Problems

The main problem with most existing localisation protocols is their notion of distance values between nodes. Most earlier work can be classified into three groups, depending on their assumptions about distance values:

- Distance data assumed to be "perfect"
- Distance data assumed to be "approximate"
- Distance data assumed to be "worthless" (range-free techniques)

The motivation behind range-free techniques - that available distance data is worthless - is partially correct, in that the current assumptions about distance data available on a sensor node make it practically worthless, but discarding it entirely gives much worse results. Additionally, most range-free techniques are dependant on a very even distribution of nodes. In this sense, they are even more locality dependant than most sensor network algorithms (we looked at locality dependance earlier in Section 1.2.3), and without very careful deployment they will exhibit even further sub-optimal results.

We realised that the problem here was another example of an abstraction problem. Specifically, all distance data available to sensor nodes is derived from other parts of the stack, which provide particular output values that are assumed to have a many-to-one (sometimes one-to-one, but the same issues occur) translation function to a distance value. That abstraction away from the original output values to a single distance value is the problem. Indeed, much simulation work (in which most localisation research is currently done, due to the difficulties and expense of setting up a physical large-scale repeatable localisation scenario at this time) discards the notion of the underlying hardware entirely, and simply provides a distance value (often with random errors added) directly to the localisation algorithms. Practically all localisation algorithms that use distance data in fact assume that a distance values feed is automagically available to them.

We decided to step back to the generalised notion of a "data source" - a piece of hardware and/or software that provides data that would have previously been used for distance calculations. This preserves the abstraction away from particular items of localisation hardware - making algorithms hardware-independant - while reducing the level of data loss. Other terminology we used in our thinking about this problem is "local node" - the WSN node that contains the "data source" and is attempting to determine distance measurements to a "remote node"; and the notion of "relative location" i.e. the location of the remote node relative to the local node.

4.3 Probability maps

An abstraction away from potentially many differing data sources is required, but a single distance value removes far too much data. We needed a better notion of a data source, that discards less information and preserves the critical differences between differing localisation hardware. We settled on the notion of a multivariate continuous probability distribution (a "map" of probabilities) for the true distance as our improved abstraction. The idea is that any particular point in the probability space for a particular input value from the data source represents the probability that the remote node is in fact located that far away from the local node.

The simplest possible probability map for a data source is the earlier assumption that it mapped to a single distance value. This is shown in Figure 4.1, and is known as a degenerate distribuiton. In this case, the figure shows that the distance value derived from the data source is 10.

A slightly more complicated example is in Figure 4.2, which shows a probability distribution centered around 10, with a gaussian distribution. One way to read this figure

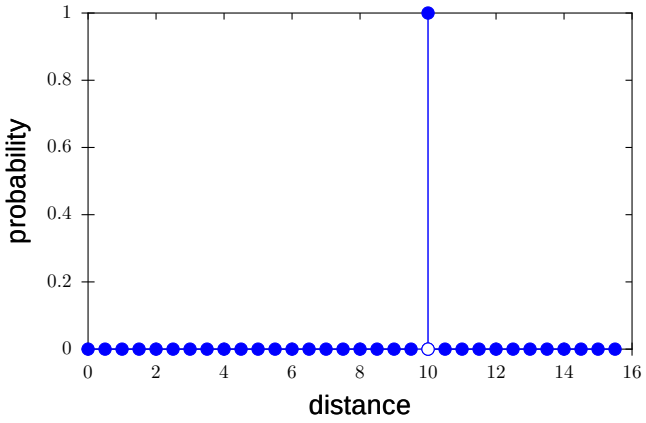


Figure 4.1: Degenerate single-value probability distribution

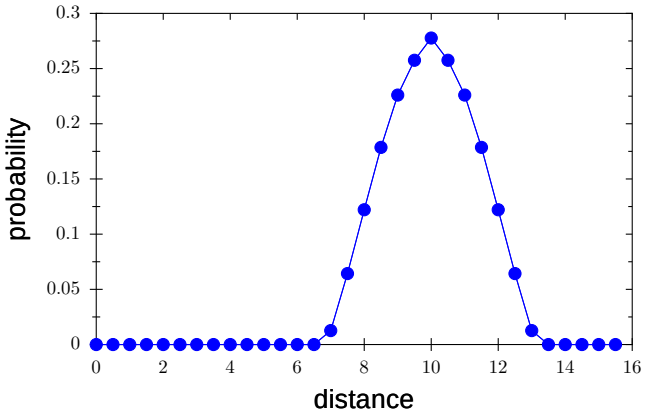


Figure 4.2: Gaussian drop-off

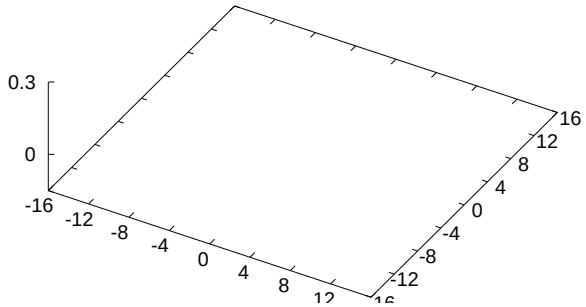


Figure 4.3: Gaussian drop-off expanded into 2-D space

is as “distance is probably 10, but there is a chance of some small variation around that value”.

Both of these examples assume that the mapping between input data and distance is the same no matter what the direction of the remote node is from the local node. In fact, Figure 4.2 can be looked at a simplified version of Figure 4.3, where Figure 4.3 shows the same gaussian drop-off centered around the distance value 10, but this time a full map around the local node is shown. As opposed to the earlier models that provide a mapping between distance and probability, a 2-D model of this nature provides a mapping between relative locations and probability. Notably, a relative location can be converted back into a distance value, but not the other way around.

4.3.1 Model choice

All of the models we have shown so far are very simple examples. The models that would be used on a node depend entirely on the hardware platform, both in terms of processing/memory usage and dependance on other hardware. Larger models require more processing and memory (and more power because of that), and the trade-off between the gains of a better model v.s. the resources used must be evaluated for each application. Additionally, if for example, orientation data is available, then a full mapping between relative locations and probability values can be built. Otherwise, if no orientation data is available, then only a distance to probability mapping can be used. Also, information about the local environment and its effect on the radios may be available.

One problem with the use of models is that there is often a significant difference between the probability model, and the actual probability of a remote node being in a particular position given the provided data source value. This is due both to any static effects that are not considered when building the model, which may include both details like a complete model of the radio, orientation data from both local and remote nodes; and what other nodes are doing that may interfere with the data source (e.g. also sending a signal on the radio).

This difference becomes more dangerous as the probability values for any individual point increases, as in effect we would then be returning towards the trivial model of Figure 4.1. Conversely, setting all values too low will cause there not to be a significant distance between very unlikely points and very likely points, making later decisions based on the models difficult.

We therefore recommend the use of relatively simple models, both because the trade-off between increased complexity and increased processing cost tends to grow exponentially; and because more complex models will tend to be trusted more than simple models, which because of the factors not taken into account (especially regarding the physical environment directly around the node, which is rarely a known factor on sensor nodes), will tend to be a bad decision.

4.3.2 Working with models

In many cases for data sources (e.g. radios) the manufacturer of the component will provide information on expected patterns for the source (radiation patterns in the radio case), but given variations in component manufacturing - especially with the cheap hardware typical to sensor networks - local calibration may help to improve data models. Whitehouse and Culler [141], as well as Balzano and Nowak [11] looked at the issues around calibrating data models, and what could be done to improve data models on a per-node basis.

Statistic-based Localisation [121] was the first work with probability models and localisation. Sichitiu and Ramadurai only looked at RSSI and did not examine the possibility of other models, but other error models could have been potentially used with their algorithms. Their core idea was that given a series of probability maps for anchor nodes, these models can be combined to calculate a larger probability map, and then find an estimated location for a node based on this map. Figure 4.4 shows a visualisation of an example map, and Algorithm 1 has more details about how they generated maps.

Statistic-based localisation has three main problems however:

- Knowledge of the complete area in which nodes will be deployed is needed to create the probability maps.
- The large amount of computation required to create a complete map (both for generation of all the points that make a single map, and to merge all the points from multiple maps together), increasing as the total area in which nodes are deployed gets larger.
- All nodes need to be at one-hop distances from the anchors (achieved in [121] by using a moving anchor with a very dense path). This is required because of a lack of a method for distributing anchor information received by one node to another, and so only anchor nodes can send their distance information.

4.4 Refined Statistics

The core approach of Statistic-based Localisation was good, but it was massively inefficient (both in terms of processing and memory requirements) and would not work without dense anchor node deployments. We made a number of changes to Statistic-based Localisation, which significantly improves the basic algorithm, making it a more useful algorithm for applications with limited resources (i.e. WSNs). We call the improved algorithm Refined Statistics-based Localisation (or RSL for short). In this section, and in Algorithm 2, we detail the improved algorithm that we used in our experiments.

Simulations with our refined approach have shown significant reductions (in the order of magnitude range) to the required processing for performing statistical localisation over the earlier work, as well as improving the generated location information in situations with non-total anchor information coverage, making possible a wider range of applications.

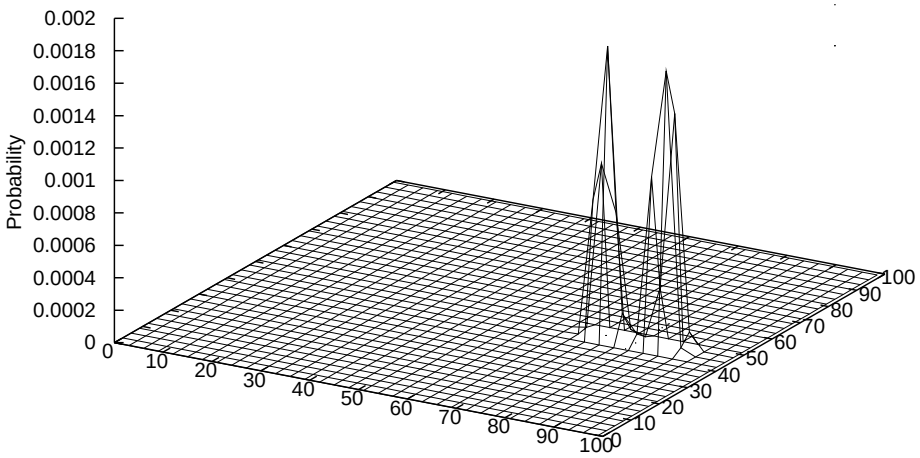


Figure 4.4: Visualisation of a complete local probability map

Algorithm 1 Statistic-based localisation [121]

1. Initially, the local probability "map" is set to a constant value across the entire sensor grid, as all locations are considered to be equally likely at the start of the algorithm.
   $PosEst(x,y) = c \quad \forall (x,y) \in [(x_{min}, x_{max}) \times (y_{min}, y_{max})]$

2. Incoming anchor information is processed as follows:

   (a) The incoming anchor location is used to create a "constraint" function on the possible locations of the node
   $PDF_{rssi} \equiv N \sim (EstimatedDistance_{anchor}, RadioRangingVariance)$
   $Constraint(x,y) = PDF_{rssi}(distance((x,y),(x_{anchor}, y_{anchor})))$
   $\forall (x,y) \in [(x_{min}, x_{max}) \times (y_{min}, y_{max})]$

   (b) The node applies Bayesian inference to its current map to generate an improved map
   $NewPosEst(x,y) = \frac{OldPosEst(x,y) \times Constraint(x,y)}{\sum_{x_{min}}^{x_{max}} \sum_{y_{min}}^{y_{max}} OldPosEst(x,y) \times Constraint(x,y)}$
   $\forall (x,y) \in [(x_{min}, x_{max}) \times (y_{min}, y_{max})]$

3. Finally, the weighted average of all of the data in the map is used to calculate the estimated position of this node
   $(\hat{x}, \hat{y}) = (\sum_{x_{min}}^{x_{max}} \sum_{y_{min}}^{y_{max}} x \times PosEst(x,y), \sum_{x_{min}}^{x_{max}} \sum_{y_{min}}^{y_{max}} y \times PosEst(x,y))$

Algorithm 2 Refined Statistic-based localisation

Abbreviations used here:
TL = Top-Left corner of a bounding box, BR = Bottom-Right corner of a bounding box, R = Radio Range of the nodes. We assume that the x-axis goes from negative to positive in a left-to-right direction, and that the y-axis goes from negative to positive in the top-to-bottom direction.

1. Initially, the bounding box for a node is set to $[(-\infty,\infty)\times(-\infty,\infty)]$.

2. As (pseudo-)anchor information comes in, the bounding box for the local node is intersected with the existing bounding box (see Figure 4.5 for examples of bounding boxes, including a diagram of this step in Figure 4.5b)
$NewBox(TL,BR) =$
$[(Max(Anchor_{TL_x}-R, OldBox_{TL_x}), Max(Anchor_{TL_y}-R, OldBox_{TL_y}))\times$
$(Min(Anchor_{BR_x}+R, OldBox_{BR_x}), Min(Anchor_{BR_y}+R, OldBox_{BR_y}))]$

3. Once information from at least two (pseudo-)anchors have been received, and the minimum waiting period since the last incoming anchor has passed, then initialise the local map to a constant value
$PosEst(x,y) = c \quad \forall(x,y) \in BoundingBox$
and then each of the (pseudo-)anchors that the nodes has received is processed as follows:

   (a) The incoming anchor information is used to create a "constraint" function on the possible locations of the node (where PDF is a function outputting the probability map value for a particular distance)
   $Constraint(x,y) = PDF(distance((x,y),(x_{anchor},y_{anchor})))$
   $\forall(x,y) \in BoundingBox$

   (b) The node then multiplies each value in the map by the constraint function to generate an improved map
   $NewPosEst(x,y) = OldPosEst(x,y)\times Constraint(x,y) \quad \forall(x,y) \in BoundingBox$

4. The location on the map with the highest probability is determined (this is the most-likely location for this node)
$(\hat{x},\hat{y}) = maxarg\{PosEst(x,y)\,|\,(x,y) \in BoundingBox\}$

5. Finally, the map is normalised to provide an externally-usable probability value
$NormConstant = \sum_{BoundingBox_{TL_x}}^{BoundingBox_{BR_x}} \sum_{BoundingBox_{TL_y}}^{BoundingBox_{BR_y}} PosEst(x,y)$
$FinalPosEst(x,y) = PosEst(x,y)/NormConstant \quad \forall(x,y) \in BoundingBox$
(this works because the bounding box always has the property that the probability that the current node is within the bounding box is 1)

If a node receives more (psuedo-)anchor data later on, then the map from step 3 can be re-used, adding only the new anchors since the last time the algorithm was run. Alternately, if the memory of the local node is limited, then the map can be discarded entirely, and the entire set of calculations needs to be re-run for each new (pseudo-)anchor.

4.4.1 Bounding boxes

If a node has received a position estimate from an anchor then it knows it is in radio contact with that anchor, and so therefore it must be within radio range of that anchor. So, we can limit the space of possible locations for that node to a circle centred on the anchor's location with radius equal to the radio range. For practical purposes (significant speed improvements) we use a bounding box rather than a circle, with each side equal to 2*radio range, and the anchor in the centre (Figure 4.5a). (The basic concept of bounding boxes has previously been analysed in [122], but not in combination with statistic-based localisation.) This results in a larger region, but we still have the guarantee that all feasible locations for the node are located within the box, while keeping the box size to a minimum. This currently assumes a circular radio model, but for radios with non-circular transmission spaces, we can calculate the minimum box that contains the entire possible transmission space, and so be still able to use this methodology.

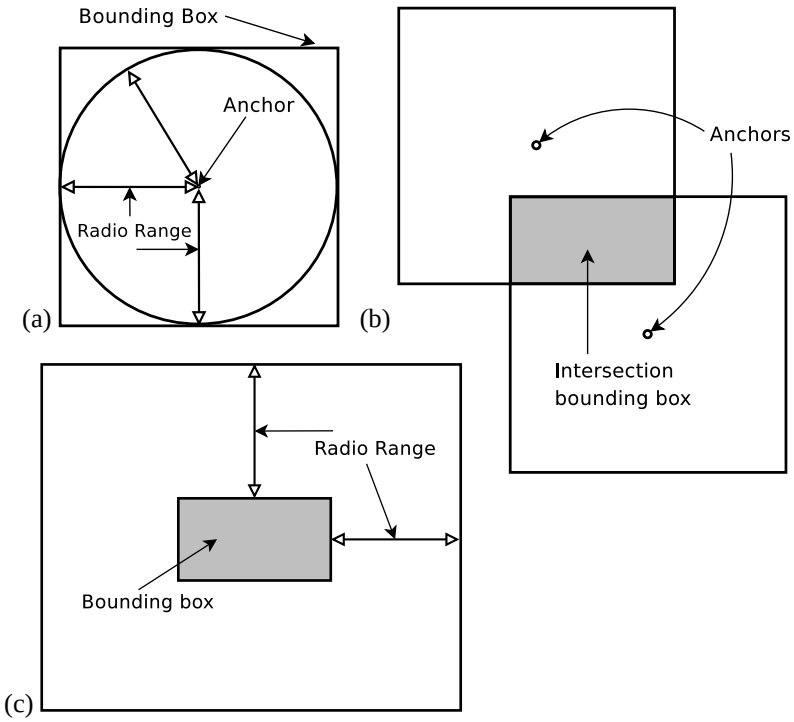


Figure 4.5: Bounding Boxes

When a node receives location information from an additional anchor, it knows that it must be within the bounding boxes for both anchors. Therefore, we can reduce the bounding box for the node to the intersection of both of these boxes (Figure 4.5b, and Algorithm 2, step 2). A bounding box is defined by two points, its Top-Left and Bottom-

Right corners. Note that the probability visualisation in Figure 4.6 only shows a partial grid (as opposed to Figure 4.4 which shows basic Statistic-based localisation, and uses a complete grid). This partial grid is the section of the complete sensor grid corresponding to the bounding box for this particular node.

Experimental results for testing the reduction in the size of the calculated sensor grid, show an average reduction in the number of required calculations by a factor of 8 when we use bounding boxes. Also, with the additional optimisation of not doing calculations for the nodes with the largest bounding boxes, we could improve this result further. For example, by discarding nodes with bounding boxes where $(width * height) > (3 * RadioRange)$ on the basis that they have too little data to be usable, we reduce the overall calculation load by an additional factor of 3.

4.4.2 Thresholded broadcast

To get around the problem of needing anchors within one-hop of the sensor nodes, we perform a limited broadcast of calculated node location information - limited by only broadcasting if we exceed a minimum probability threshold for the quality of our location information (currently set in our implementation to 0.003). The node effectively acts as an additional "pseudo" anchor, but with two changes from normal anchors.

Firstly, location information is broadcast with a confidence value (gained from the local probability map), and the error model used by nodes receiving this information will be scaled accordingly, as shown in Algorithm 2, step 3a with the use of $Confidence_{anchor}$ in the generation of PDF_{rssi}. This confidence value is a weighting value for use in the statistical models i.e. a node with confidence 1.0 (an anchor node) will have twice the effect of a node with confidence 0.5.

Secondly, with pseudo anchors, the bounding box is broadcast as well, and the box used by receiving nodes is not just a square centred on the node (as for anchors), but a rectangle equal to the bounding box size, plus radio range in each direction (Figure 4.5c). This is because the bounding box contains all locations the pseudo anchor could possibly be in, and so increasing it by the radio range creates a box in which nodes that can hear this pseudo anchor could possibly be located. This box will be larger than a box generated from an anchor node, because the location information is less accurate. However, this larger box may still be useful to other nodes in reducing their bounding boxes, and hence reducing the amount of computation that they need to perform.

Nodes that have position information, but do not exceed the probability threshold are considered "bad". These nodes have some position information, but either the information is insufficient, or it is of too low a quality to be fully usable. These do not broadcast their location information to other nodes.

One additional scenario that uses pseudo-anchors is when we have location information from another system (e.g. GPS) and this data is inaccurate. We can then treat this inaccurate anchor as a pseudo-anchor, with an appropriate confidence value and bounding box depending on the incoming data. Refined statistic-based localisation does not actually specifically require accurate anchors, but simply some sources of initial localisation data to initialise the algorithm.

In our experiments comparing this multi-hop method with the original single-hop method, we see a similar average error in the locations of the good nodes, but a 38% average increase in the number of good nodes.

4.4.3 Symmetry problem

There are a number of situations where we will have multiple points that have equally high probabilities (or certainly very similar, and within the bounds of statistical error). One of the most instances of this problem is when we have multiple anchors in a straight line. As the distributions of the anchor data cross over equally on both sides of the line, a pair of possible good points will be created, each one equally far away from the line, but on opposite sides. Figure 4.6 is an example of this, showing the local probability map for a node with this particular problem.

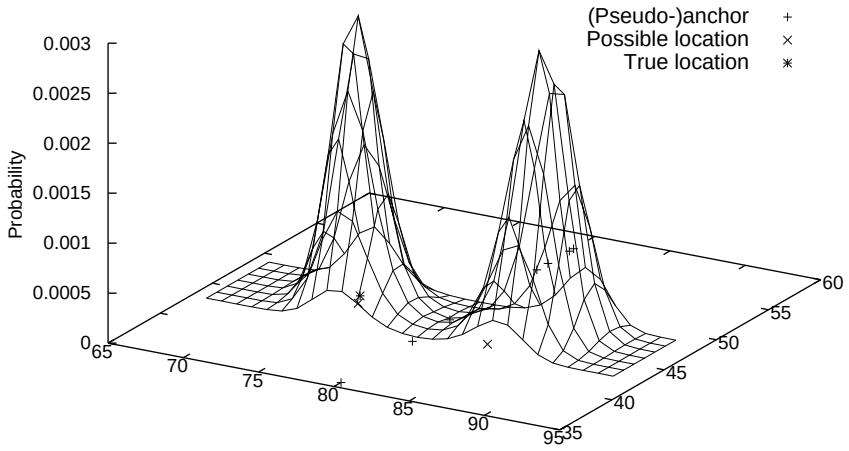


Figure 4.6: Equal points

The broadcasting of derived-anchor data from nodes with good location data reduces the symmetry problem, as this creates additional pseudo-anchors, allowing the possible locations for the node to be "pulled" in the direction of the correct point. However, in the event that the paths of the anchors is a straight line, and that there are insufficient "good" nodes in the local area to broadcast pseudo-anchor information, then the problem still occurs. One solution is to avoid deploying nodes in a straight line - random distribution is best, but curved deployment paths will also help a lot.

A node can determine whether or not it is likely to have multiple possible positions, based on its local probability map, by calculating the average of all of the anchor locations the node knows about (weighted according to their confidence values), and seeing how much each anchor's location differs from their average location in each separate axis. This allows the node to check whether its known anchors are mostly arranged along a straight line, or whether they have a more varied path. If there is a significantly greater

total difference from the average point in one axis than another (indicating a mostly straight path), then the node will also test the other possible good points. These can be found by taking the averaged anchor location, then looking at the points that are on the opposite side of the average point from the calculated most-likely location for this node. An average of the most-likely location and the other possible points (weighted according to their individual confidences) becomes the node's estimate of its true location.

If the sum of the confidences for the most-likely point and the best of the other candidate points, divided by a scaling factor, is above the standard threshold for transmission of the calculated location, then we transmit both locations. The scaling factor varies according to the degree of difference between the two confidences i.e. how good the second confidence is compared to the first.

$$ScalingFactor = 2 \times \frac{confidence(Best) + confidence(SecondBest)}{confidence(Best)}$$

If the second point is similarly confident to the first, then the scaling factor will be proportionally greater, but it always satisfies the condition $2 \leq ScalingFactor \leq 4$.

If the node decides to transmit its current guesses, then the confidences for both points are transmitted, and the node is treated as two separate nodes by its neighbours, one at each of the two possible points, but each with a reduced confidence (compared to the calculated confidence for the point).

4.4.4 Heavy data-processing

One downside of statistic-based methods is the amount of data processing required to calculate the local maps. The bounding boxes reduce this significantly (a factor of 8), by eliminating many regions that this node can not be located at. For best results, there should be a waiting period for a short amount of time (e.g. 5 seconds) after the last piece of anchor information has been received, before calculating the local map, in order to work with the smallest possible bounding box. This will slow down the calculation of this node's location, but given that it is necessary to re-calculate the data if we receive another anchor, this can reduce the amount of redundant calculations significantly. The waiting period should be calibrated such that if we have not seen a new anchor for that amount of time, then we are unlikely to receive more anchor information in the near future. Good values for this would be at least as large as the interval between broadcasts of the mobile anchor.

The energy costs associated with statistic-based localisation are higher than for most localisation techniques, due to the large number of probability calculations required (Lateration being a notable exception, due to the use of matrix multiplications), but this additional cost is in most cases a one-off initialisation cost. Simulation results show an increase in processing time for refined statistic-based localisation over deterministic techniques [86, 115, 116] by approximately a factor of two; an increase from ˜5 to ˜10 seconds, results will vary depending on bounding box size and CPU, this is for a typical node CPU [133] and an average bounding box taken from experimental testing. This order of processing time is not an unacceptable start-up cost for a long running application,

given the significant improvements in the derived location information. The probability calculations can also be performed by many nodes at the same time without additional costs, as opposed to other localisation techniques requiring large numbers of radio messages (which would exhibit increased numbers of packet collisions if several nodes are transmitting radio messages at the same time). Refined Statistic-based localisation has been deliberately optimised towards reduced radio traffic with this aim in mind.

An additional optimisation that could reduce the data processing cost is the alteration of the grid size of the calculations. When the probabilities for a region are calculated, this is done at discrete intervals, resulting in a grid of probabilities for a region. The distance between points in the current implementation is fixed in size, but this could be varied on a per-node basis, depending on the size of the bounding box. Larger bounding boxes would increase the point distance, and smaller boxes would decrease it, creating an approximately equal point count (and therefore processing time) for all nodes, while allowing nodes with small bounding boxes to have more accurate estimates than they achieve currently. The point count could also be varied at the application level, to allow for application-specific accuracy requirements.

Refined Statistic-based localisation could also be further improved by the limiting of Step 5 in Algorithm 2. Currently, we generate the normalised map for all locations in a node's bounding box. However, we then only use a small subset of this data. An additional reduction in processing time could be gained by only calculating normalised probabilities for the most-likely location and for the additional possible locations needed for the solving of the symmetry problem (Section 4.4.3).

4.5 Mobile anchors

RSL will work with only a small set of anchors, but the results become much better with greater numbers of anchors. Therefore, in this section we look again at how initial anchor information can be provided to an ad-hoc sensor network, given typical resources for target application scenarios.

4.5.1 Anchor distribution

Most methods for providing location information to a sensor network start with adding additional localisation hardware (e.g. GPS) to a small percentage of the nodes in the target area. These anchor nodes will initially gather accurate location information on their own, and then transmit this information to their neighbouring nodes. This approach has a number of major faults:

- Most localisation algorithms based on "spread anchor" scenarios rely on the anchors being evenly distributed across the sensor network. Unless special care is taken to make sure of this, or a very large percentage of the nodes are anchors, then this is unlikely. Given a small anchor percentage (as in most proposed applications), there is a high probability that there will be regions of the sensor grid that

have insufficient anchors, leading to problems in attempting to localise nodes in those regions.

- Anchor nodes are generally more expensive (because of the additional hardware requirements), which creates a difficult decision regarding the balancing of the application requirements between having improved accuracy (lots of anchors) and reducing the overall cost of the network (few anchors).
- The additional anchor hardware is often only useful during the initial phase of the network setup, and is then mostly surplus to requirements. An anchor node may also have a reduced operational lifespan due to the power drains of the localisation hardware.

There have been some attempts to fix these problems (Adaptive Beacon Placement [17] for example), and there are partial fixes, but a better approach is to look at other ways that location information can be distributed rather than the use of static anchor nodes.

4.5.2 Mobile anchor scenarios

Mobile anchor scenarios [121] are an alternate approach, resolving a number of the problems with the spread anchor scenarios. This approach uses a single, large anchor capable of moving along a path. This large anchor could be carried by a car or a person for example. The intention is that this larger anchor will have effectively unlimited power (i.e. can transmit as many messages as needed) because it is intended to be more easily accessible than the individual sensors, and so replacing the anchor node's batteries is less of a problem than replacing batteries in the sensor nodes.

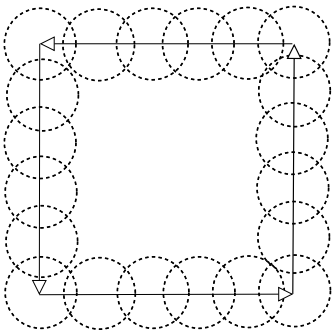


Figure 4.7: Example mobile anchor scenario

As the mobile anchor moves, it broadcasts its location at regular intervals (either every few seconds, or after it has moved a short distance from its last broadcast location), thus creating a series of "virtual" anchors, as in Figure 4.7. Each circle represents a position where the mobile anchor has broadcast its current location.

4.5.3 Real-world applications

To see how mobile anchor scenarios map onto various applications, we looked at the structure of these applications, and saw how we could better utilise the already available resources. The main area of interest regards the method for the distribution of the sensors. A number of different methods have been proposed, varying from the manual placing of individual nodes, through to the dropping of nodes from a plane. These methods can be

grouped into two categories depending on the distance from the object that is placing the nodes to the location that the nodes are being placed.

The simplest scenarios are when the distance is less than the nodes' radio range (ideally much less). In this case, the placing object itself (be it a person or a car) is the mobile anchor. This can be achieved by combining an anchor node with the placing object (either carried by the person, or attached to the car). It can then broadcast its location information as it places the nodes, thus providing a path that passes near all of the nodes.

More complicated are the situations where the nodes are far away from the placing object, for example when dropping nodes from a plane (especially from a high altitude, or when trees or other obstacles are likely to block radio signals from the placing object). One solution to this problem is that the plane could also drop one or more small robots fitted with localisation equipment, in addition to the sensor nodes. These robots could travel along a semi-random path around the sensor grid (with constraints to keep them near the grid), providing location information to the sensor nodes as they move around.

4.5.4 Advantages

There are several main advantages of mobile anchor scenarios:

- Instead of many anchor nodes (and having to make the trade-offs regarding how many) we have effectively many anchor nodes, but for the cost of only a few anchor nodes (one per placing object). The anchor infrastructure is therefore "there when you need it, not when you don't". All of the sensor nodes should have similar lifetimes, without the additional power drains that would occur if some of them were also anchors for the network.

- In the complicated scenario with the use of mobile anchor robots, the cost of the scenario does go up from what would be possible with more simple scenarios. However, the robots could also be fitted with additional sensors (above and beyond what would be fitted to normal nodes), so that once they have finished providing location information to the network, they can be moved to locations where interesting events are happening to gather more detailed information. The possibility of very simple (and cheap!) sensor nodes coupled with larger robot-mounted sensor arrays would provide a cost effective methodology for detailed data gathering without requiring every node to have a large sensor array.

- In the event that the initial anchor path is not sufficient to provide good location information for all of the sensor nodes, we may (depending on the application) be able to do on-the-fly improvements in bad areas. The equivalent solution [17] for standard anchor scenarios would involve placing additional anchor nodes, at additional cost, but with mobile anchors we can simply move the mobile anchor near the inaccurately located nodes. These can be discovered using iterative location techniques working from the closest known other nodes.

4.6 Results

Using the Positif simulation framework for localisation algorithm testing [72], we have performed a series of comparison tests between RSL, and three deterministic localisation techniques (Euclidean [86], Hop-Terrain [115] and Multilateration [116]), using a mobile anchor scenario in all cases, and with a variety of ranging errors between nodes.

In each case, all of the algorithms have been tested with the same set of data, and each result is the average of 10 runs of the simulation with varying random-number seeds. The ranging error is modelled as a Gaussian distribution, with the mean as the actual range, and the range variance as a percentage of the radio range. The internal model of the refined statistic algorithm in all cases is set to a Gaussian distribution with the mean as the incoming range information, and the estimated range variance set to 20% of the radio range. In all scenarios, there are 226 sensor nodes randomly placed, with a uniform distribution, within a square area. The mobile anchor is modelled as a formation of 111 "virtual" anchors within this sensor grid. The grid has a size of 100x100, and the radio range is set to 14 providing the nodes with an average connectivity of 19.

There are three different mobile anchor scenarios being considered here. The first is a "square" formation, with a mobile anchor moving along a square path situated approximately 1/5th of the sensor grid width from the edge of the grid at all times. The second is a "cross" formation, testing what might happen with two separate mobile anchors, one moving from the the top-left to bottom-right, and the other moving from bottom-left to top-right. In both cases, the start points are situated 1/5th of the sensor grid width from the edges of the grid. The straight lines of these two topologies have been deliberately chosen to cause difficulties to RSL. The third topology is a "wobbly" square, taking the square formation locations as a base, and then moving the anchors by a random amount (maximum distance of 2, uniformly distributed) away from the initial location to provide a less straight-line path.

Figures 4.8 (square), 4.10 (cross) and 4.12 (wobbly) are visualisations of the individual node locations for a set of example experiments that we have performed using RSL. The nodes marked with a "•" are anchor nodes, the others are sensor nodes; the ones marked with a "*" are good nodes, nodes marked with a "+" are bad nodes, and the "△" nodes have no position data at all. Lines attached to nodes show the path from a node's true position to where it thinks it is. The longer the line, the less accurate the estimated position. Note that, in general, RSL does a good job of classifying the nodes into good and bad ones, but occasionally generates both false positives (good nodes with long lines) and false negatives (bad nodes with short lines). These anomalies generally occur outside the area directly covered by the mobile anchor. Since the node classification is largely correct, applications should be able to exploit that knowledge to their advantage.

In figures 4.9, 4.11 and 4.13, we show the average accuracy of the good nodes for all of the algorithms. For RSL, we also have bad nodes, so we also show the accuracy for a weighted average of both good and bad nodes. Figures 4.14, 4.15 and 4.16 show the average percentages of positioned nodes in each of these cases. Note the poor coverage (generally less than 50% of the nodes obtain a position estimate) for the square and cross

topologies, which shows the problems induced by non-uniform anchor distributions in combination with the symmetry problem for straight line topologies.

In most cases RSL has the lowest percentage error in its ”good” positions. Euclidean only outperforms it under ideal circumstances (i.e. no range errors); in all other cases (error variance $> 5\%$) RSL provides (much) more accurate position estimates. In general, localisation algorithms can trade-off accuracy for coverage [72]. RSL, however, combines high accuracy with reasonable coverage. For low error variances, RSL has similar numbers of good nodes as the Hop-Terrain and Multilateration algorithms, only at higher values RSL starts to classify more nodes as being bad. The combination of the RSL good and bad nodes however, gives a comparable level of error to the other algorithms, but with up to a doubled number of positioned nodes.

The “square” topology was chosen as a typical example of a simple mobile anchor scenario, which could have been implemented by for example a mobile node attached to a car driving around a square-shaped building. These x/y-axis aligned paths can be detected by the equal pairs heuristic (Section 4.4.3) in some cases and compensated for accordingly. Despite the problems still occurring due to the straight paths, RSL is still capable of getting reasonable results.

The “cross” topology was designed to attempt to break the current implementation of RSL, as the equal pairs heuristic does not work as well with diagonal paths. However, although RSL has less accurate results for the cross topology, all of the other algorithms also do badly as well. We would therefore not recommend the use of the “cross” topology for use in mobile anchor applications.

The “wobbly” square topology is an example of a topology that should be easier for the localisation algorithms, as the significantly lower errors for this topology shows. One example case where we would expect to see this sort of topology is where the mobile anchor is attached to a soldier patrolling the perimeter of a base. The significantly better results with this topology over the straight-line topologies is why it is recommended to avoid straight lines with the mobile anchor paths.

In all of the experiments the internal model of RSL has been set to a Gaussian distribution with variance as 20% of the radio range. The results for that particular variance of the actual errors are not much better than for other variances of that magnitude. Note therefore, that we can get good results even when the actual ranging information model is significantly different from the internal model of the algorithm. It is important to try and get the internal model as similar as possible to the actual model, but as these results show, good data can be acquired even if the internal model is inaccurate.

4.7 Related Work

As previously mentioned, Sichitiu and Ramadurai [121] did the initial work with statistic-based localisation and mobile beacons. They however required an order of magnitude more processing time, plus a far higher anchor coverage density to achieve similar results to refined statistic-based localisation. Sun and Guo [130] also looked at probability models, but without exploring the sources of such models in much depth.

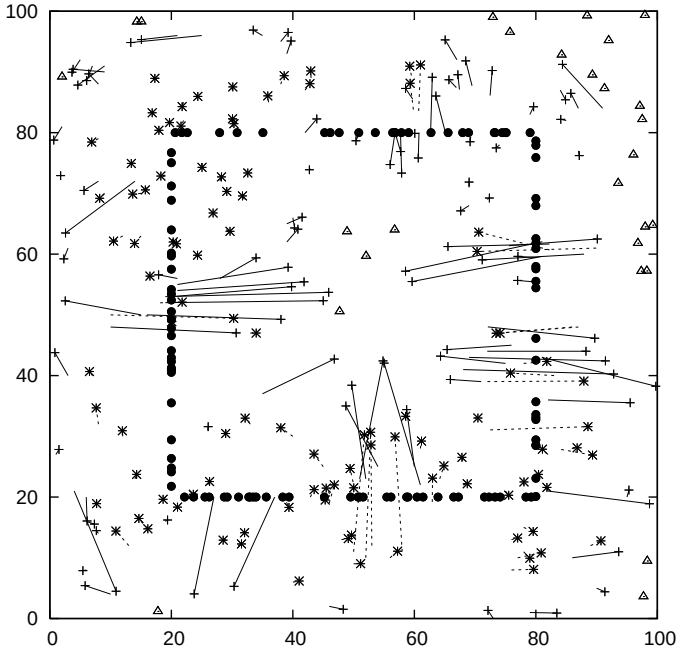


Figure 4.8: Square topology, 20% range error variance

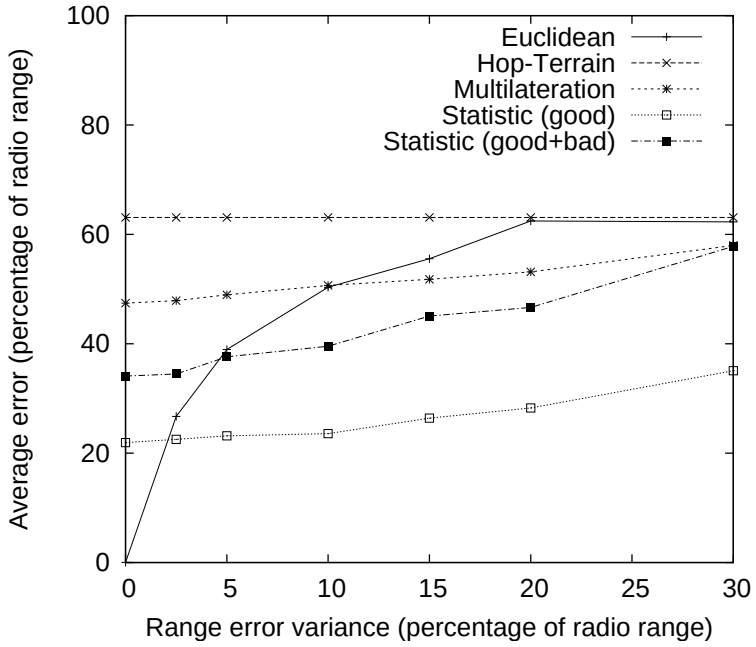


Figure 4.9: Square topology accuracy

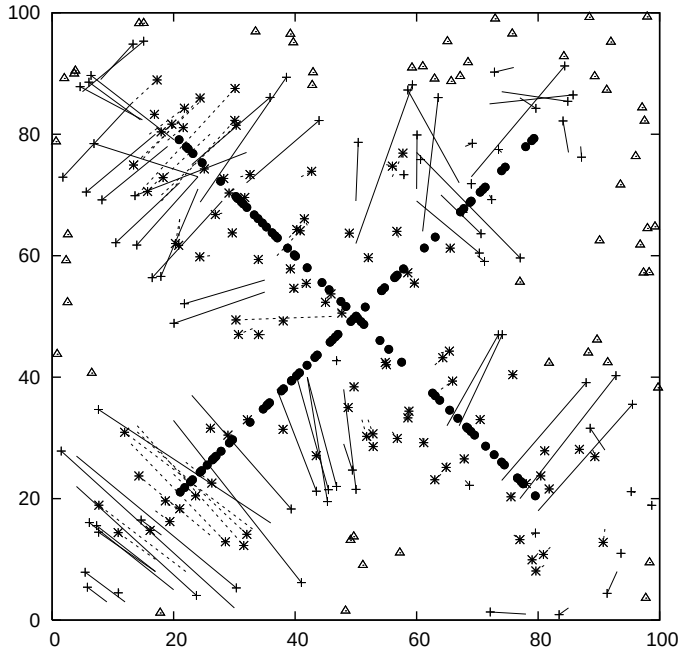


Figure 4.10: Cross topology, 20% range error variance

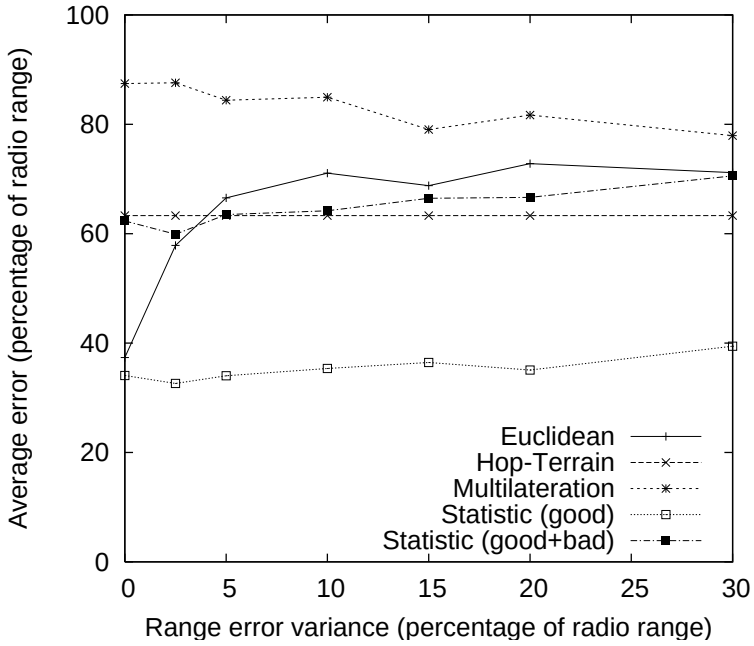


Figure 4.11: Cross topology accuracy

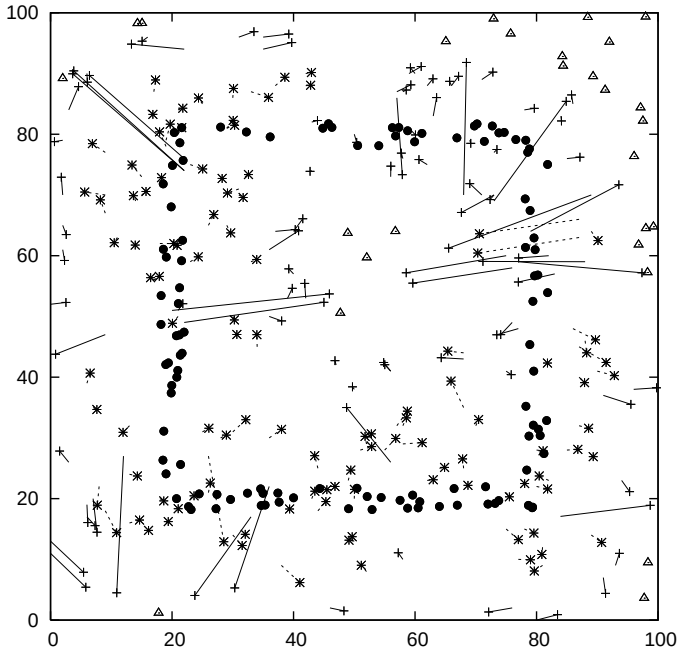


Figure 4.12: ”Wobbly” square topology, 20% range error variance

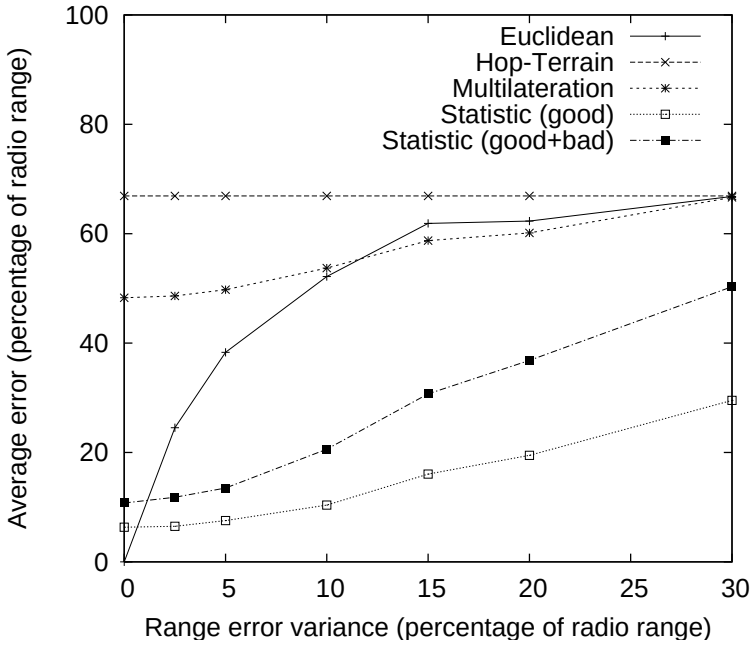


Figure 4.13: “Wobbly” square topology accuracy

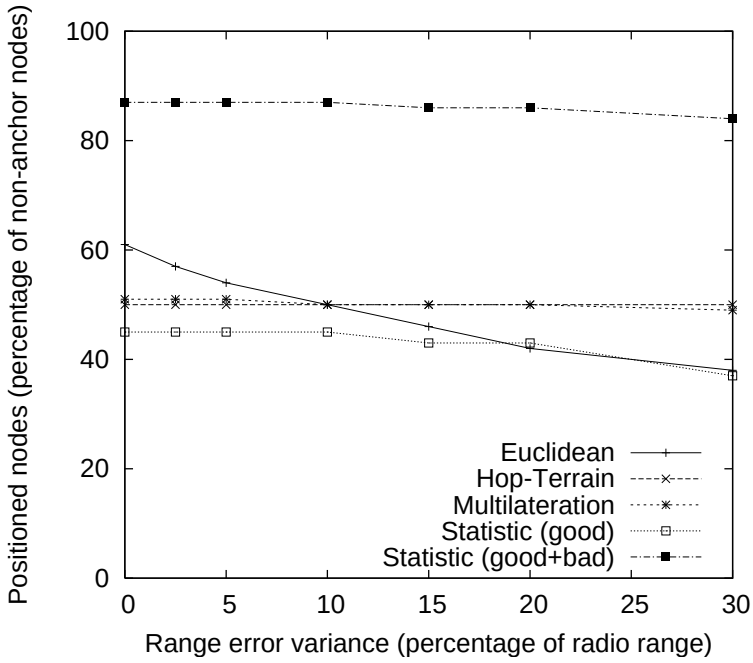


Figure 4.14: Square topology coverage

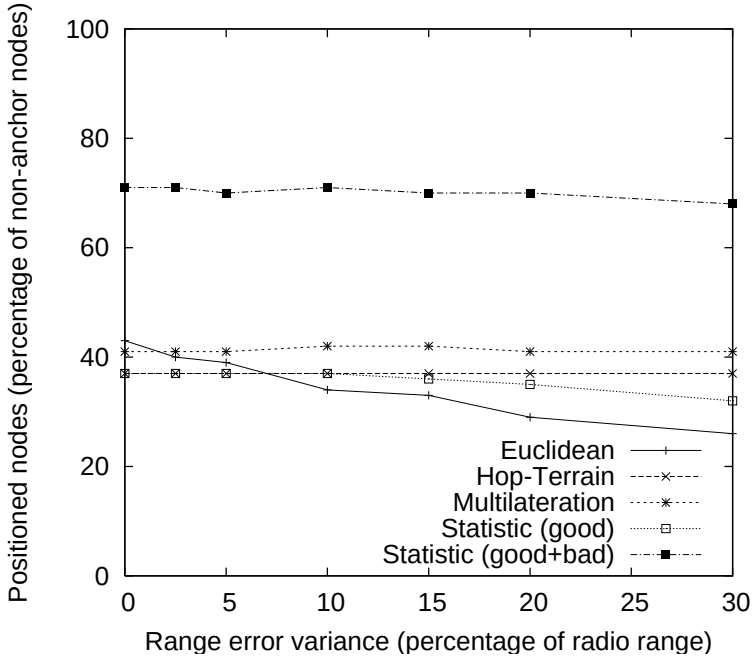


Figure 4.15: Cross topology coverage

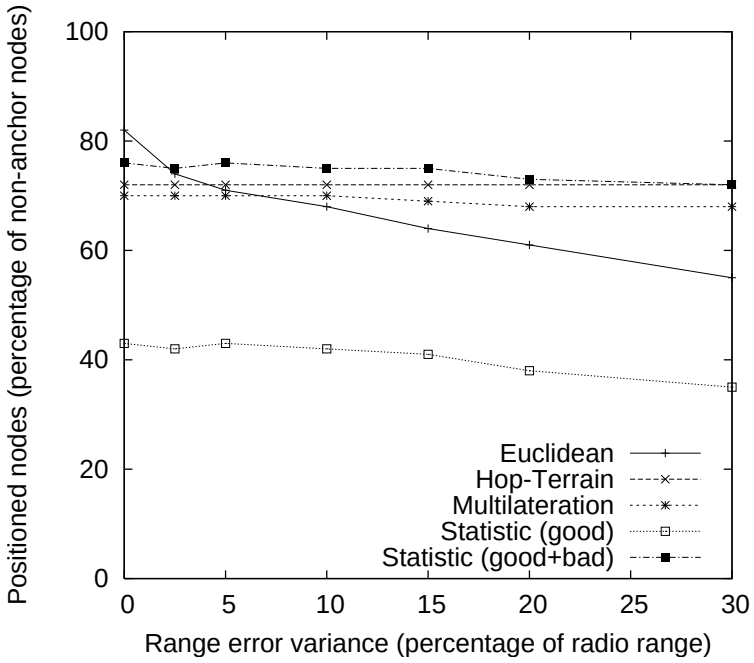


Figure 4.16: "Wobbly" square topology coverage

Ssu et al. [127] also worked with mobile anchor points, but their scheme was heavily dependant on circular radio ranges, which is often not the case with sensor node radios [153].

Galstyan et al. [36] did some earlier work using bounding boxes, with additional optimisations in the area of "negative information" i.e. if two nodes can not communicate with each other, they are assumed to be out of range with each other. Bounding boxes have the assumption that a node is certain to be somewhere within them, but given the significant likelihood of bad links (two nodes that are in radio range but cannot communicate) in the real world due to a variety of possible problems (e.g. objects in the way), this will cease to be the case if we use negative information. Results from [**?**] indicate that even without such obstacles, bad links still occur in a significant percentage of cases.

Doherty et al. [28] proposed a method for constraint-based localisation, including work with bounding regions. However, their techniques required several orders of magnitude more processing power than the methods proposed here and so they used centralised computation of their algorithm - rendering it unsuitable for large sensor networks due to the overheads of exchanging information with a central node. Their solution to the centralisation problem using a hierarchical distribution of the problem would still require much larger energy/computing resources than present on current node hardware, but now the additional capacity would have to be evenly spread across the network, reducing the feasibility of this technique even further. They also did not achieve significantly better results than the distributed algorithms demonstrated in this chapter.

4.8 Conclusions

We presented here an approach that can provide good location information, even with non-uniform anchor distributions and considerable inaccuracies in the incoming ranging data. We were able to do this by re-examining the abstraction of distance between nodes as the core building block for building localisation algorithms, and moving to probability density functions instead of single values as the abstraction for distance estimation. Probability density functions provide a better way to abstract away from the raw sensor data used to generate distance-related information, rather than the simplified approach with simple distance estimates.

Refined Statistic-based Localisation provides a good solution to the problem of localisation even in small, resource-limited sensor networks. We showed that we can calculate accurate position data for a high percentage of the sensor nodes in a network, improving the quantity of positioned nodes in sensor networks both versus simpler statistic-based methods and deterministic localisation methods.

All of this has been tested using mobile anchor scenarios, which we have shown to be a realistic and usable method for the distribution of anchor data, as well as a cost-effective one - both in terms of energy costs for the sensor nodes of the network (as the mobile anchors are separate from the deployed nodes), and in terms of the necessary hardware required to create the sensor network. Getting rid of the errors in sensor measurements is hard to do well, but that is the price of gathering data from the real world. With statistical approaches, we have shown that it is possible to work around these errors, and derive good location information. Statistical approaches are somewhat more computationally expensive, but given the significant improvements in the location information, and that the computational expense results in a reduced level of required radio traffic during the localisation process (which increases the capability of other nearby nodes to do radio-dependent work efficiently during the localisation process), we believe that the trade-offs are worth it.

4.8.1 Future Work

In the future, we hope to expand on our work here to attempt to further improve the location information that can be gathered, by integrating more accurate models of various ranging sensors, and also testing to see whether a combined model from several sensors may improve accuracy. More work also still needs to be done on making it easy to build localisation algorithms on standard node hardware platforms, as very little work is done with real hardware.

Recent work [60] has shown that better bounding boxes can be derived by altering the transmission power to provide multiple options for the size of a box, and so this would help to reduce the size of the bounding boxes.

Chapter 5

Motion

In this chapter: We further re-examine the abstractions developed in Chapter 4 (probability maps and bounding boxes), look at differential probability maps, and build new protocols that can do motion detection both with (Portmanteau) and without anchor nodes (Adumbrate), but without requiring motion-detection hardware.

As we discussed in Chapter 4, many possible applications have now been thought of for Wireless Sensor Networks (WSNs), and a significant number of them rely on location information in order to perform their designated function. This is because the main purpose of a WSN is information gathering, and gathered data is only useful if you know what it applies to.

The range of viable localisation techniques depends heavily on what node hardware is available. At one end of the scale, if every node has accelerometers, GPS, and an array of accurate ultrasound sensors, then localisation is quite simple. Alternately, nodes can have no hardware designed for motion detection or localisation at all, and only RSSI data from a radio to give limited ranging information. Unfortunately, most node hardware is of the latter type. Additionally, most existing work [17, 47, 86, 115, 116, 121] generally deals with static networks, and detecting when a network is no longer static with minimal additional hardware requirements would be of considerable use.

With WSN localisation, nodes with additional hardware are referred to as "anchor" nodes i.e. they have a reliable source of information about their location. Many localisation techniques rely on anchors, and on the assumption that anchor nodes are uniformly distributed among a uniform distribution of non-anchor nodes. Given the small percentage ($<$10% in most scenarios currently postulated) of anchors within a large collection of non-anchors, and the aim that sensor networks are eventually intended to be easy to distribute for non-computer scientists, this assumption can not be relied on for many application scenarios.

Most of the content in this chapter has been published as "Adumbrate: Motion Detection with Unreliable Range Data" by Tom Parker and Koen Langendoen at the Fourth International Conference on Networked Sensing Systems (INSS 2007) [3]

One piece of information that would be very useful is motion information - if a node has not moved between its initial deployment and the time it is fully localised, then we know that all data gathered up until that point was from a particular location. If it has moved, information on the approximate amount of motion may help decide whether the data can still be treated as located at a particular point (with a particular level of location accuracy). We therefore need to be able to detect motion even without anchors.

Another major problem within WSN localisation techniques is acquiring accurate range information between pairs of sensor nodes. There is generally some error in the ranging information (as we noted and discussed in Chapter 4), which motion detection algorithms must be aware of and be able to work with.

5.1 Detecting motion

Despite the potential usefulness of motion detection for sensor networks, especially given their locality dependance (see Section 1.2.3), it has been a neglected topic. Most work with moving nodes tends to focus on anchor nodes (e.g. Mobile anchor scenarios from Section 4.5), or with robots [94, 113, 135]. In both cases, it tends to be assumed that the moving nodes/robots either have an external source of localisation data (e.g. GPS), and therefore can calculate motion information where needed; or that they have motion sensors available to them (e.g. an accelerometer). Unfortunately this is not true for most typical sensor nodes, and so this limits what can be done for motion detection using techniques that rely on this extra hardware being present on all nodes. This creates a percieved difficulty in doing motion detection on typical sensor nodes, to the extent that it is not even considered as an option, despite the potential uses of such data.

We therefore decided to further explore what could be done in the area of motion detection without changing the hardware profile for standard WSN applications, as we felt that the standard reasoning regarding what is available to a typical sensor node was flawed, and that there were possiblities for doing motion detection by taking apart the abstractions that describe standard thinking about node hardware.

In this chapter we focus on two scenarios for motion detection - what can be done with just basic nodes (no localisation hardware; just RSSI); and what can be done with minimal quantities of additional hardware on a limited set of nodes (anchor nodes). The first scenario is in line with our goals of zero changes, but the second requires additional hardware on a subset of the nodes. As this additional hardware was already part of the requirements for Localisation protocols, we felt that exploiting this hardware without adding further hardware requirements was an acceptable compromise.

With only basic nodes we are limited as to what we can do, but some information can still be gathered. In the situation with a limited set of anchor nodes, we still may well have the same problem as with just basic nodes, as with low percentages of anchor nodes, a given basic node may well have no communication with anchor nodes. One solution to the lack of additional anchor nodes is that the anchors may well be mobile (Section 4.5), and so even if a basic node has no current communication with anchor nodes, gathering some information before communication is established with anchor nodes may help de-

termine earlier location data. Anchor-less situations are likely in the early stages of some mobile anchor scenarios, especially when the placing object is far away from the locations where the nodes are being placed, and so we need to be able to do motion detection both with and without anchors.

Our particular focus here is on allowing smarter decisions in limited motion scenarios for localisation algorithms designed for static networks, and limiting the problems that moving non-anchor nodes can cause to stateful localisation techniques. In general, stateful localisation tends to break if motion occurs, but by being aware of motion, protocols can discard/revise localisation state to avoid this happening.

Unfortunately, most methods for detecting movement of nodes can not tell the difference between moving nodes and malicious nodes (nodes that are sending bad data). Malicious nodes are hard to deal with - with a large enough amount of effort and/or nodes, a malicious intruder can potentially break an entire network. However, for most non-military sensor network scenarios, the chances of a malicious intruder are very low, whereas motion is likely. We are therefore going to concentrate our efforts on detecting motion, and leave the problem of dealing with malicious intruders for more advanced systems.

5.2 More Probability maps

Motion is intrinsically linked to localisation, as if a node moves it changes from being at one location to being at another. Therefore, our first intuition was that looking at data from localisation techniques (Chapter 4) would give us some insight as to how to do motion detection on sensor nodes.

In Section 4.3, we looked at the standard mapping functions from sensors capable of providing data that could be converted into range values, and concluded that instead of providing point values, a probability map for possible inter-node ranges was a more useful abstraction from the possiblities for sensor data. In a similar manner, we considered some options for probability mapping for motion detection.

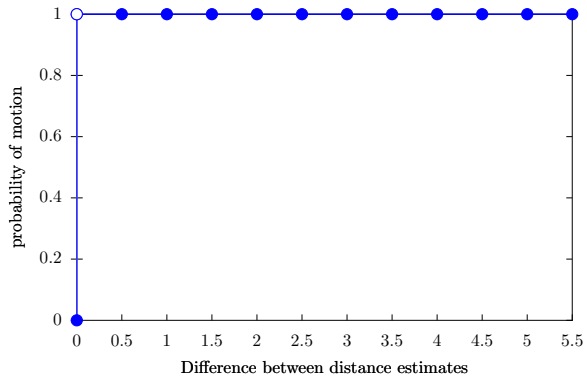


Figure 5.1: Trivial case

We ideally wanted a probability map in a similar manner as for distance values, providing a mapping between an input value from a sensor and the probability that a node had moved. For dedicated motion detection hardware (e.g. an accelerometer), such a map would look like Figure 5.1, a trivial degenerate case with all values > 0 indicating motion. For our use cases, however, such hardware was not available, and we were likely to have to work with inaccurate distance measuring hardware.

As we only have distance measuring hardware to work with, we need to look at the difference between a distance measured at one point in time and a distance measured at a later point in time, i.e. compare two probability maps. We were initially concerned that the likely patterns for probability maps shown so far, along with other studies regarding the inaccuracy of RSSI [108, 145, 153] (which is the most likely available source of data for deriving distance numbers) would give significant variations in the maps for multiple sequential readings of the RSSI values between a pair of static nodes. In other words, we expected to see a lot of variation in the RSSI values, and so therefore lots of perceived motion even when the nodes were not moving.

We further investigated the sources of RSSI variation, and found that in general the variability in RSSI between a pair of static nodes a fixed distance apart should be significantly lower than the variability between random pairs of nodes that are the same distance apart e.g. if nodes *A*, *B* and *C* are all 10 metres apart, then multiple readings from a single pair of nodes will show less variability than multiple reading from different pairs of nodes, despite the distances being the same. This is a result of several of the primary sources of RSSI variation (node orientation, sender characteristics, receiver sensitivity) being identical for all of the multiple readings from the single pair of nodes, but will be different for the different pairs, despite the distances being the same. Figure 5.2 indicates a typical probability map for differences between two RSSI values for a pair of static nodes.

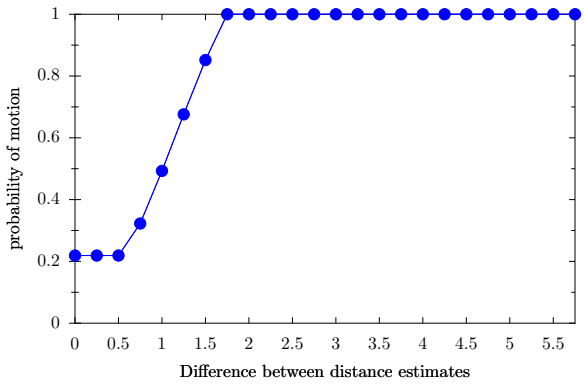


Figure 5.2: Example differential probability map

However, if the environment changed, then the RSSI values changed as well. Additionally, there were occasional apparently random spikes in the RSSI values even without

changes to the environment. This meant that although we could generate maps similar to Figure 5.2, they would have more outliers and a generally lower trustability than the source probability maps. We therefore needed a better way to filter out the outliers and other problems caused by a changing environment. Notably, this is the situation that we faced during the Localisation work - the probability maps on their own were not very useful as a distance measure, but were a useful raw data source for more accurate methods.

As we do not have good location data, we cannot use RSL for merging probability maps. In fact, the lack of location data for any nodes stops most methods of merging the maps directly. We instead looked at the idea of mass-spring models, using the original distance probability maps to generate the force equations for the springs.

5.3 Adumbrate

When we refer to radio range in this section, we are using the maximum possible radio range between a pair of nodes, including any "gray area" [**?**] effects. The techniques here have been influenced by [51] (specifically their gradient descent algorithms for mass-spring systems in the design of Section 5.3.2.4).

Mass-spring models are a good example of the experiential metaphors mentioned in Section 1.3.2 - a mental model based on physical experiences. They are based around the idea of a series of point masses joined together by a series of springs. For node motion detection, we build a mass-spring model detailed in this section, referred to as Adumbrate, where the nodes are the point masses, and the springs represent the estimated ranges between a pair of nodes.

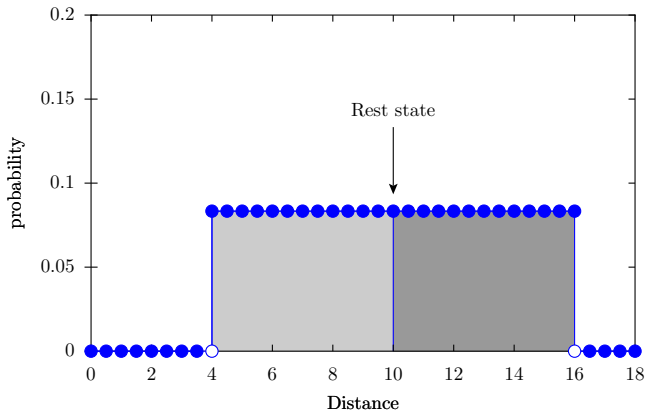


Figure 5.3: Example Hooke's law probability map

Normally, springs in mass-spring systems would be modelled according to Hooke's law of elasticity, where the force on a particular spring is equal to the difference between the rest state and the current elongation of the string multiplied by a string constant, *k*. Instead, we use the probability maps for the ranges between the nodes to generate an

estimated force on a spring based on the cumulative probabilities for a node being further away or closer than the current range estimate.

For example, Figure 5.3 shows a probability map for an example spring that actually follows Hooke's Law. The maximum radio range in this case is 16, and the rest state is 10, because there is exactly the same area in each of the grey areas. In other words, if the length of the spring is 10, it is equally probable that the spring (= range between the nodes) is actually shorter or longer than that length, and so the force on the spring at that length is zero.

Adumbrate moves away from trying to generate physical locations for the nodes, as we do not have location data, but will provide us with topological data suitable for determining information about the probable motion of the nodes. In order for this to work, at least a subset of the pairs of nodes need to have stable ranging values, both in the sense of the nodes being physically static between the first and second measurement of the data source (RSSI) and the values from the data source need to be almost identical. If there is not a stable subset of nodes - either because of too much motion, lots of change to the environment (which will effect the RSSI values) or unstable data readings - then reliable motion detection cannot be done using Adumbrate.

5.3.1 Mass-spring model

In Adumbrate, the energy $U_{A,B}$ of the spring between a pair of nodes A and B with a current range of $R_{A,B}$, a maximum radio range of M, and a probability map defined by a function $f(x)$ can be given by the general-purpose formula

$$U_{A,B} = k(\int_0^{R_{A,B}} f(x) - \int_{R_{A,B}}^{M} f(x)) \tag{5.1}$$

where k is a model-dependant spring constant.

For example, the two grey areas from Figure 5.3 are equal to $\int_0^{R_{A,B}} f(x)$ and $\int_{R_{A,B}}^{M} f(x)$ respectively for a value of $R_{A,B}$ equal to the rest state. This works both with probability maps that are defined by a function (e.g. gaussian distribution), and those that feed from a table of data derived from real-world experiences.

One problem with the use of the general-purpose formula is that repeatedly deriving values for it can be a relatively expensive operation, and for most sensor nodes this can be a problem. The problem is amplified by the fact that this function will be called frequently by other operations as we will see later on. Therefore, for models that are function-derived, it is often better to create an approximation function which provides a model-specific energy function for lower computational cost.

In our experiments, we worked with node-node ranges assumed to be $\sim N(m,v)$ (gaussian errors around a most-likely value of m with variance v). For this model, the energy function is:

$$U_{A,B} = k\frac{|R_{A,B} - m|}{v} \tag{5.2}$$

Both classes of energy function should be equally usable, provided the same function is used for all nodes within a particular system.

5.3.1.1 Links

In order to know which pairs of nodes need to have springs inbetween them, we define the concept of some nodes being "linked" to others. A link between a pair of nodes is defined as one of two possibilities, either

1. A and B can communicate directly i.e. A and B have a known value for the measured radio range between them. A is therefore a neighbour of B and vice versa.

2. $R_{A,B} < \textit{radio range}$, but A and B are not connected using the previous rule. In this case the link distance is defined as the radio range, and the $U_{A,B}$ result is scaled by the probability of a broken link (i.e. $U_{A,B}^{broken\,link} = U_{A,B} * \textit{BrokenLinkProbability}$) as given from experimental data. Values for the broken link probability will be approximately in the 0.1-0.2 range. A and B in this case are not neighbours, but they are linked.

A link creates a "force" due to the spring that pushes/pulls the node towards a more accurate location. For a given node A, we can calculate the force F_A on that node using

$$F_A = \sum_B F_{A,B} = -\sum_B (A \hat{\rightarrow} B) U_{A,B} \tag{5.3}$$

where $A \hat{\rightarrow} B$ is the unit vector from A to B and A and B are linked.

5.3.2 Local co-ordinate systems

The node that is running Adumbrate is referred to as the "root" node. In order to do motion detection, we first need a method to build local co-ordinate systems.

5.3.2.1 Reference node placement

Firstly, we need to gather range data (estimated values and variances from the radio model) from the root node to its neighbours, and also query the root's neighbours for range data to their neighbours, giving us a topological map for all of the root's 1- and 2-hop neighbours. We can then place the root node, and one of the root node's neighbours. In order to define a local co-ordinate system, we need reference points. The root node is declared as being located at $(0,0)$, and we also require a second "reference" node to define the x-axis for this system.

We need a node that is highly connected to the root node's immediate neighbours, in order to reduce the quantity of calculations we need to perform later on. Therefore, the selected reference node will be one of the 1-hop neighbours of the root node, and we select it using the following rules in order

1. Highest number of root-transitive links (i.e. for a given node, the number of its neighbours that are also neighbours of the root node).

2. Highest number of neighbours.

3. If we still have >1 possible nodes, pick one randomly (lowest node id is a suggested method).

We now also declare this selected neighbour as being initially located at $(m,0)$ where m is the measured distance to the neighbour. As this always makes $U_{Root,Neighbour} = 0$, this is currently a minimum energy configuration of the positioned nodes.

Once we have the reference node and root node placed, we then move onto the other nodes.

5.3.2.2 Initial placement

Working from these initial two nodes, we can now start to find initial locations for the other nodes. We can place all nodes that have two neighbours in the already placed set of nodes, using those two neighbours (A and B, referred to as the "parent" nodes of our new node) and the ranges between them to place our new node C. In some cases, we will have chosen parent nodes that are unsuitable for placing C, and in these cases Adumbrate will fail the sanity tests specified below. If this is the case, we then proceed to check other possible parent node pairs for suitability.

For a node C with already placed neighbour nodes A and B, and A and B are neighbours of one another, we may be able to calculate an inital location using A and B as parent nodes to C. Using the measured values for all of the inter-node distances, we start by calculating $\angle BAC$ from the law of cosines.

$v = \frac{R_{A,C}^2 + R_{A,B}^2 - R_{B,C}^2}{2R_{A,C}R_{A,B}}$, $\angle BAC = cos^{-1}(v)$

Sanity assumption: $|v| \leq 1$

Using a line D, parallel to the x-axis but through A, we then calculate the angle of $A \hat{\rightarrow} B$ to D

$n = \frac{A_x - B_x}{R_{A,B}}$, $z = sin^{-1}(n)$ where z is the angle of $A \hat{\rightarrow} B$ to D

Sanity assumption: $|n| \leq 1$

We can now calculate two possible values of θ (= angle of $A \hat{\rightarrow} C$ to D), using $\theta = z \pm \angle BAC$. We then have two possibilities for C's co-ordinates using the two values of θ and $C = (A_x + R_{A,C}cos(\theta), A_Y + R_{A,C}sin(\theta))$. These are shown on Figure 5.4 as C and C'. We choose the initial location of a node with the minimum amount of force (as defined in 5.3.1.1) given the current set of placed nodes.

In some cases we will fail the sanity assumptions, and have to test with other pairs of neighbour nodes. Once we have placed all of the nodes that have a valid pair of placed neighbours, we then work on the remaining nodes.

5.3.2.3 Placing remaining nodes

If we have remaining unplaced 1-hop neighbours of the root that do not have 2 neighbours in the set of already placed nodes, then we can repeat the process for selecting a

reference node (as in Section 5.3.2.1, but using only non-positioned nodes as possibilities), and place this newly selected neighbour at $(-\frac{\sum_p^{placed} p_x}{n}, -\frac{\sum_p^{placed} p_y}{n})$ i.e. an averaged location directly opposite the current set of placed nodes, which is the most likely location for this remaining unplaced node. We now return to the process of placing additional nodes that have two neighbours in the "already placed" set, and if necessary keep repeating this sequence of processes until all the 1-hop neighbour nodes are placed.

After placing all of the 1-hop neighbours, if we still have unplaced 2-hop nodes with 2 placed neighbours, but for all possible pairs of placed neighbours *A* and *B*, *A* and *B* are not neighbours of each other, then we use the calculated locations for a pair of neighbours to work out the distance between them. The calculated distance is then used temporarily for the placement steps in Section 5.3.2.2. This is less accurate, but will still give us a reasonable first guess for the location of a node.

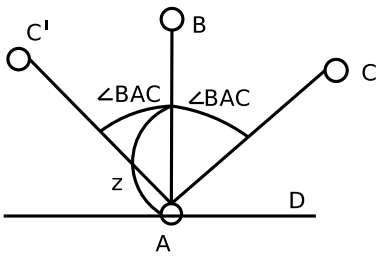


Figure 5.4: Placing C

If there are still unplaced 2-hop nodes, without at least 2 placed neighbours then these 2-hop nodes must have 1 placed 1-hop neighbour (by the definition of a 2-hop node as being connected to a 1-hop node, all of which have now been placed), then we place the 2-hop neighbour at

$(\frac{p_1^x(r_1+r_2)}{r_1}, \frac{p_1^y(r_1+r_2)}{r_1})$ where $r_{\{1,2\}}$ is the root→1-hop and 1-hop→2-hop measured ranges respectively, and $p_1^{\{x,y\}}$ is the x- and y-coordinates of the 1-hop neighbour.

Placing the 2-hop neighbour further along the line of the 1-hop neighbour provides a reasonably likely initial position, without the need for extensive calculations on the full set of placed nodes.

5.3.2.4 Topology optimsation

The locations for the nodes are now further refined. Refinement is necessary because our initial configuration does not take into account all of the links between nodes when we are placing them.

The total energy of the system in a particular configuration is

$Energy = \sum_{A,B} U_{A,B} \quad A,B \in placed\,nodes$
and there exists a link between *A* and *B*

An optimal topology for a mass-spring system is when the total energy of the system reaches a pre-defined minimum value (ideally zero, but in practice this will often not be possible to achieve). We may not be in this state after the inital placing, as we did not take all of the link information into consideration initially. We therefore need to further refine our location data.

The location of each node A is refined, firstly for the 1-hop neighbours, then the 2-hop neighbours. For 2-hop networks, this makes sure that a node's parents will always be evaluated before the node itself. A is refined as follows:

1. If A has an parent node that switched to its alternate location during this round of the algorithm, then recalculate A's location and alternate location according to the previously specified initial placing algorithm (Section 5.3.2.2).

2. Otherwise

   (a) Calculate A's current force F_A, with Equation 5.3 on page 99.

   (b) If A has an alternate location, which is a valid location given the communication links to this node i.e. all direct links to A are within radio range of the alternate location, calculate the force for the alternate location as well, and if the magnitude of that force is smaller, A is moved to the alternate location.

   (c) Update A's current estimated location
   $A \leftarrow A + F_A T$
   where T is an arbitrary constant controlling the rate of convergence.

These steps are repeated until a minimum energy state is reached, or until the reduction in energy from one state to the next drops below a pre-defined limit (or the energy increases). One possibility for improving the speed and accuracy of this process is to choose a value for T that is proportional to $Energy$, allowing for rapid reductions initially, reducing the motion as we progress towards the minimum energy state. Other techniques such as simulated annealing [62] could also be applied to select suitable values for T.

5.3.3 Motion detection

Now that we can build a local co-ordinate system, motion detection is possible by comparing a local co-ordinate system generated at one moment in time (LCS_1) by a node, to another generated system by the same node at a later point in time (LCS_2). We require at least 2 nodes common to both systems (which may or may not be neighbours), in order to be able to use this information, otherwise we cannot work out which way the node moved.

For each pair of nodes which we will designate A and B, using the LCS_1 system co-ordinates for A, B and our root node (marked as I), as well as range data from LCS_2 for our root node relative to A and B, we can calculate the possibilities for the location of the LCS_2 root in LCS_1 (designated as K) using

$$(K_x - A_x)^2 + (K_y - A_y)^2 = R_{K,A}^2 \quad (5.4)$$

$$(K_x - B_x)^2 + (K_y - B_y)^2 = R_{K,B}^2 \quad (5.5)$$

Solving for K_x in terms of K_y, A and B, gives us

$$e = R_{K,A}^2 - A_y^2 - A_x^2 \tag{5.6}$$

$$m = \frac{e - (B_y^2 + B_x^2 - R_{K,B}^2)}{2(B_x - A_x)} \tag{5.7}$$

$$n = \frac{2(B_y - A_y)K_y}{2(B_x - A_x)} \tag{5.8}$$

$$K_x = m - nK_y \tag{5.9}$$

Using Equations 5.4 and 5.6-5.9 we can then solve for K_y

$$o = (A_x n - A_y - mn)^2 (m^2 + 2A_x m - e) \tag{5.10}$$

$$K_y = \frac{-2(A_x n - A_y - mn) \pm 2\sqrt{o(n^2+1)}}{2n^2+1} \tag{5.11}$$

Equation 5.11 gives us two values for K_y which we can then substitute back into Equation 5.5 to get values for K_x.

$$h = R_{K,B}^2 - B_y^2 - B_x^2 \tag{5.12}$$

$$K_x = -B_x \pm \sqrt{2B_y K_y + h - B_x^2 - K_y^2} \tag{5.13}$$

This gives us up to 4 (K_x, K_y) pairs that represent potential values for K. Results involving imaginary numbers are discarded, as they do not represent valid solutions.

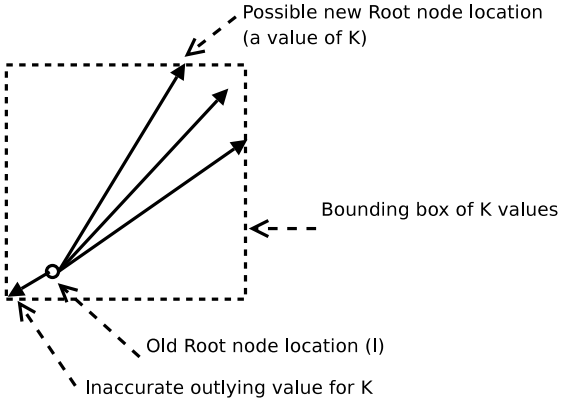


Figure 5.5: Calculating values for K

For each pair of nodes common to both co-ordinate systems (A and B), and using the LCS_1 system co-ordinates for A, B and our root node (marked as I), as well as range data from LCS_2 for our root node relative to A and B we can calculate the set of possibilities for the location of the LCS_2 root in LCS_1 (Figure 5.5).

We now have a set of up to 4 possible locations for K which are checked against the measured $R_{K,A}$ and $R_{K,B}$ values. The values that have correct ranges (at most two of them, by standard geometrical theory regarding the intersection of two circles [139]) are valid locations for K, and we choose the closest to the existing root node, as the movement between separate invocations of this algorithm should be minimal.

Each of the valid K locations represents a "motion vector" (MV) for our root node. We can calculate values for MV using the locations of K as the vector between I and K, as in the event of no changes, $K = I$, and I is at $(0,0)$ by the definition of I being the origin of the local co-ordinate system. The average of the values for MV is the assumed motion, and the maximum K values in each direction gives us a bounding box whose area is proportional to the inaccuracy in our K measurement.

5.3.4 Results

We performed a series of experiments to test Adumbrate, starting from a randomly generated set of "true" node locations, using 226 nodes in a 100x100m area, with a radio range of 14m, giving an average connectivity of approximately 12.

Experimental tests [108] have shown that the change in the error between consecutive measurements for the range between a pair of static nodes, will be significantly smaller than the error between the measured ranges and the true range. This is because many of the sources of range inaccuracy (reflections, batteries running down, low-quality radios, etc) should be relatively stable between one range measurement and the next. We therefore setup our experiments to mimic this, by taking the topology and ranging information from the "true" locations, and adding some gaussian distributed noise to the ranging data (mean equal to the "true" range, variance at different levels for different experiments). This "noisy" ranging information was then used to generate a local co-ordinate system (Section 5.3.2). We then moved the root node by a random amount (uniformly random direction, distance depending on the experiment). For all the links not connected to the root node, we changed their "noisy" ranges by a small random value (mean equal to the original "noisy" range, variance at different levels for different experiments), and for the links connected to the root node, we re-generated new "noisy" ranges according to the true ranges for the new root node location (noise generated with the same parameters as the first local co-ordinate system). This second set of "noisy" data was then used to generate another local co-ordinate system, and the two were compared as per Section 5.3.3.

For all of the experiments, the results are specified as percentages of the radio range, and are averages of 20 runs of a particular set of parameters, using a different random seed each time. Figures 5.6, 5.7 and 5.8 show the results with inaccuracy for the non-moved links set to 0%, 5% and 10% of the original variance. The 6 lines on each of the graphs represent a variety of movements of the root node between the first and second sets of data. At 10% and 20% motion, neither altering the original error nor the second measurement inaccuracy significantly changes the results, and the percieved motion is reasonably accurate ($\pm$3%). However with greater motion ($>$20%), the percieved motion becomes increasingly inaccurate. Note that this is the motion between successive tests of

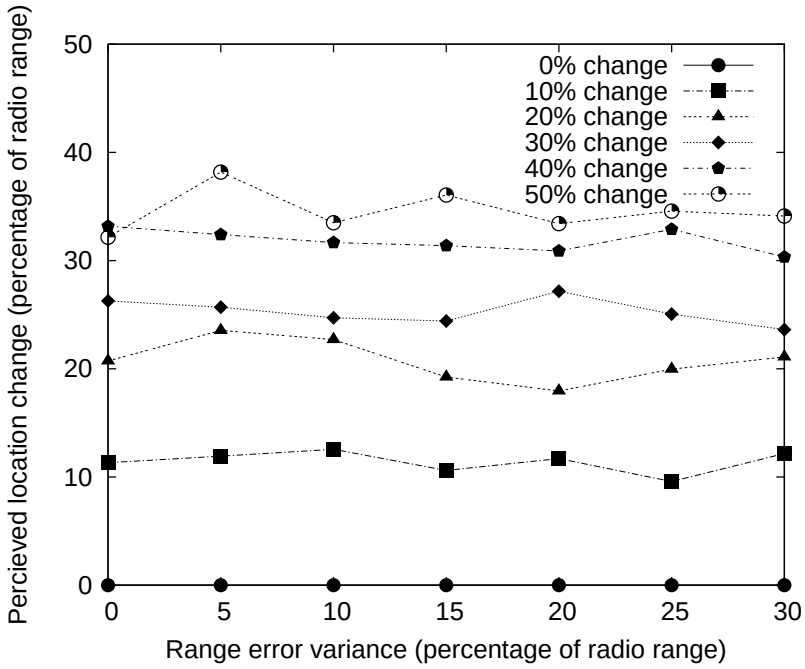


Figure 5.6: 0% inter-measurement inaccuracy

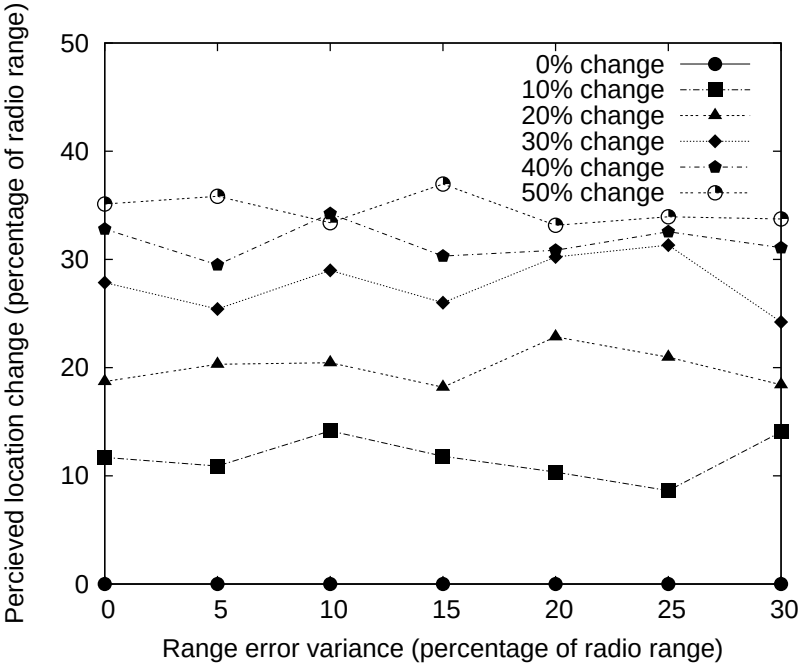


Figure 5.7: 5% inter-measurement inaccuracy

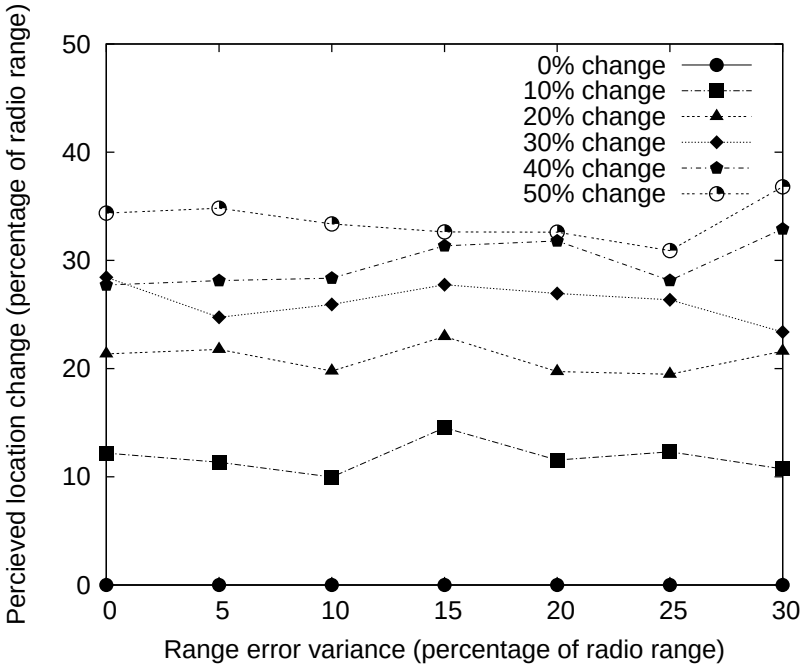


Figure 5.8: 10% inter-measurement inaccuracy

the motion detector, and so if we run the algorithm frequently enough (depending on the average rate of motion of the root node) these more difficult cases can be avoided.

The curves in all cases are relatively flat - a first guess at expected results for these experiments would assume an upwards curve in perceived motion as the error between true and measured distances increases. However, the motion detection algorithm that we are using here works with the differences between two measured distances, and as the errors for each of the two measured distances are similar, increasing the error from true distances does not significantly alter the algorithm's results.

Increasing the change in the error between the two measured distances does not change the results that much either, and this also applies with additional tests that we have done for higher values of the error change. The error values that we have used here are similar to values shown in experimental testing [108].

5.4 Moving localised nodes

We have now shown that motion detection can be done without any knowledge of the physical locations of the nodes. However, this is subject to numerous false positives, especially when a node is moving rapidly between the first and second estimates of the

ranges between nodes. In this section, we show how to detect motion with localised nodes, without false positives, and in a way that works better with more movement.

If a node has been localised, and then moves without being aware of its movement, then the node will be somewhere other than where it thinks it is. If it then broadcasts its old location data, while being at the new location, then other nodes in the network will have inconsistent information. This is only a problem with non-anchor nodes, as when anchor nodes move to a new location, they will have new location data, and in both cases their true and calculated locations are the same (to within a known degree of accuracy).

5.4.1 Bounding boxes

Anchor nodes periodically broadcast their locations, and if a node has received location data from an anchor then it knows it is in radio contact with that anchor, and so therefore it must be within radio range (where "radio range" is a maximum possible value including "gray area" [**?**] effects) of that anchor. Thus we can limit the space of possible locations for that node to a circle centred on the anchor's location with radius equal to the radio range. Bounding region information can therefore be used to sanity-check information from localisation algorithms. For practical purposes (significant speed improvements) we use a bounding box rather than a circle, with each side equal to $2 * radio\,range$, and the anchor in the centre.

We looked at this in detail earlier in relation to Localisation (Section 4.4.1), but we now focus on the motion related information that we can derive from bounding boxes.

5.4.2 Breaking the Boxes

As a consequence of the sanity condition that a node's bounding box will always contain its true location, and that any two nodes that are in communication must be within radio range of each other, bounding boxes assume another sanity condition - that the current bounding box of a node and another bounding box that it has received, and therefore wishes to intersect with, will always have a non-empty intersection.

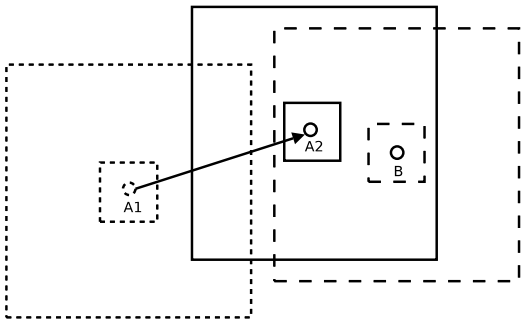


Figure 5.9: Motion example

Figure 5.9 is an example of how motion of a node can break bounding box sanity. $A1$ and $A2$ are the locations of a moving node A before and after it moves, and B is a stationary node. The inner and outer boxes around the nodes represent their bounding boxes and bounding boxes expanded by radio range, respectively. If A talks to B when it is at $A2$, and A thinks it is still located at position $A1$, then there will be an inconsistency between A's bounding box and the bounding box of B, which means that one of the two nodes must have moved. In a

number of cases we will not be able to detect motion (the maximum allowable motion before we can detect motion with absolute certainity is proportional to the size of the bounding box of a node), but in these cases we do maintain bounding box consistency, so we can still generate valid bounding boxes, although with a reduced accuracy due to the size of the boxes.

5.4.2.1 Portmanteau

When we do detect bounding box inconsistencies, we can work to correct the problem, using the algorithm described below, which we refer to as Portmanteau. If a node N receives a new bounding box from a neighbour M that would create an inconsistent situation ($Box_N \cap Box_M = \oslash$), then this tells us that either that N or M has a problem. Both nodes then check how many of their neighbours currently consider them inconsistent. If two neighbours (including either N or M) consider one of N or M currently inconsistent, then that node should recalculate its bounding box information. This is done by discarding all current bounding box data (i.e. returning the node to Step 1 of Algorithm 2 on page 77), and sending a control packet to all of the neighbouring nodes saying that any currently used bounding box information from that node should be discarded, and requesting their current bounding boxes.

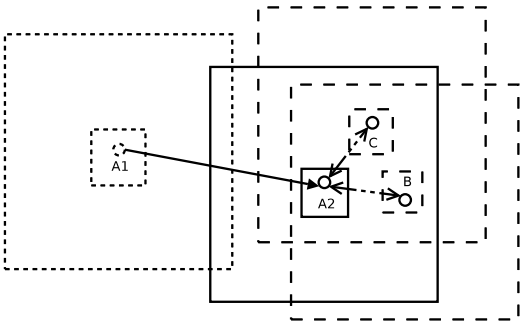


Figure 5.10: Inconsistency

Figure 5.10 shows how this could work for a node A moving from $A1$ to $A2$. It starts to communicate with nodes B and C, and there is an inconsistency between the box $A1$ and the boxes for B and C, so there is an inconsistency "link" from $A \leftrightarrow B$ and from $A \leftrightarrow C$. As two of A's neighbours consider it inconsistent, it resets its bounding box data back to the startup configuration, and sends a control packet to B and C invalidating any bounding box data they have gained from A, and requesting their bounding boxes. This would then result in a new and valid bounding box for A. Any localisation algorithms being run on the node should also possibly be notified at this point if the previously determined location for the node is now outside the new bounding box.

In many of the possible scenarios for bounding box inconsistency, the problems will now be resolved, and the node will have a new bounding box. If however, this fails, then the node should send a message to its neighbours declaring that it currently considers them inconsistent, and remain in an inconsistent state. The inconsistent node should now stay in that state until there is a change in any of its neighbours' bounding boxes, in which case the bounding box for this node should be re-evaluated to check for the resumption of consistency.

Figure 5.11: Bounding box testing

One problem here is that *B* and *C* may have previously integrated *A*'s information into their bounding box configurations, and if *A*'s information is later found to be invalid, then *B* and *C* need to be able to work out what parts of their bounding boxes are due to *A* and what are due to other nodes. In order to counter this, each node can keep a record of the bounding box for each other node, in order to be able to rebuild an accurate bounding box when one node's information is found to be invalid. If a node resets its bounding box information due to detected inaccuracies, then the node also discards the list of bounding boxes that it had stored as well. To deal with mobile situations where there are many various sources of anchor information, and the storage of every other recieved node would be impractical, then only a limited set (*N* most recent recieved boxes) are stored, in addition to the calculated box for the node in question. Discarding some recieved boxes after they have been used to improve the local box does reduce our capability to handle inconsistent boxes due to motion, but given the limited storage available to WSN nodes, this is a reasonable trade-off.

5.4.3 Results

We performed a series of experiments, testing how much motion was necessary before Portmanteau could detect inconsistencies. The nodes were scattered in a 200 x 200 box, with radio range set to 14. We varied the number of nodes to get different levels of average connectivity in the network, as well as designating a percentage of the nodes as

anchor nodes. For each simulation run, we allowed the box sizes to exchange bounding boxes as per Section 4.4.1 (without any of the limits on number of transmissions to attempt to generate the best possible bounding boxes) until they stabilised at their smallest possible values, and then started to move one of the nodes in a random direction. Each experiment was run 20 times, with varying random seeds for each configuration, and the results given here are an average of the 20 sets for each configuration.

The graph in Figure 5.11 shows the minimum motion necessary before inconsistency checking noticed motion. The minimum motion necessary for detection reduces with higher connectivity networks, as well as with increasing anchor percentages. For most scenarios, the amount of motion necessary does not in general exceed *radio range*. Additionally, all of the experiments reported a zero false positive rate i.e. no node reported as moving was in fact stationary.

5.5 Related Work

Capkun et al. [19] created an algorithm to create local coordinate systems, and a method for translating from one system to another. They then proceeded to attempt to use a network of co-operating nodes to build a Network Coordinate system (a form of local co-ordinate system where all of the nodes in a network use the same local co-ordinate system), using a Location Reference Group (LRG) of semi-stable (i.e. minimal movement) nodes as a centre for the topology. We have used an LRG-like system here, but using information from a local neighbourhood rather than the entire network. Network Coordinate systems significantly increase the amount of traffic required to setup and maintain the system over local coordinate systems, and that cost rises with the size of the network. The benefits gained via the use of this are minimal, and in most mobile anchor scenarios the situation where no anchor information is available is for a limited time only, and so cross-network protocols that could utilise a network coordinate system (e.g. source-to-sink message routing) would be better off storing data locally and waiting for anchor information before transmitting. Priyantha et al. [102], as well as Shang and Ruml [118] also looked at LRG-like systems, with similar problems regarding processing and communication costs as Capkun et al.

Krumm and Horvitz [65] did some earlier work with motion detection using RSSI. Their method used smoothed histograms of varying signal strength from Access Points (APs) in an 802.11 network to determine whether a particular node was moving. The motion detection algorithm did not explicitly use location data, but the requirement for the APs to be static allows them to be used as reference points. Our work here requires that a subset of the nodes being measured are relatively static (such that comparisons between the different local co-ordinate systems can be made), but without requiring that the currently-used static nodes remain permenantly static. If an application has permenantly static infrastructure nodes (e.g. an urban 802.11 network), other more efficient algorithms are possible, but this cannot be guaranteed for WSNs. Muthukrishnan et. al [85] also did similar work more recently, but using FFTs (Fast Fourier Transforms) to process the RSSI data instead of smoothed histograms.

5.6 Conclusions

We have shown here another way to look at localisation data for the purposes of motion detection. With standard approaches to localisation, only single range values would be available, and motion detection with that data would be very limited and significantly inaccurate. By finding new applications for the abstractions developed in Chapter 4 - probability maps and bounding boxes - much more can be done. In particular, the mass-spring model, with its representation of inter-node distances as springs in order to limit the effect of errors occuring in multiple measurements over a period of time, provided a new way to look at the problem of motion.

We have also shown that even in situations where localisation breaks down (such as anchor-less scenarios) that motion can be detected without having location data, and without requiring extra hardware. Working with inaccurate ranging data, even with the probability maps, is difficult, especially without anchors to provide sources of known sane data. With mass-spring anchor-free approaches, we have shown that it is possible to work around the inaccuracies and derive good motion information. Mass-spring approaches are somewhat more computationally expensive (compared to using motion detection hardware), but given the ability to provide motion information without requiring extra hardware or anchor nodes, we believe that this is worth it. Mass-spring approaches are also able to rapidly detect motion, but at the cost of introducing the chance of false positives. Additionally, if we do have anchor nodes, then we have shown how we can detect motion with simple methods based on bounding boxes, and reduce the false positives level down to zero.

5.6.1 Future Work

To a certain extent, all of the possible future improvements mentioned for Localisation apply here as well, given the dependancies of the techniques here on the abstractions detailed in Chapter 4. Any improvements in the probability maps or reductions in the size of the bounding boxes will improve the results shown here.

Additionally, we would like to explore integrating together data from both mass-spring models and bounding box methods into a unified “end-to-end” motion detection algorithm, incorporating data from results over the entire lifespan of an application. Mass-spring models are good at noticing relatively small amounts of motion, with some level of false positives, but less good with larger quantities of motion. Bounding box methods are bad at noticing small amounts of motion, but once motion levels exceed certain thresholds then they are much more capable of noticing it, and without false positives. Creating a system that blends the best of both would be a sensible option which would, for example, allow the bounding box methods to tell the mass-spring models which nodes have moved “too much” and so let the mass-spring models work with just the low-motion nodes.

Chapter 6

Aggregation

In this chapter: We challenge both the use of standard statistical functions for aggregation, and the notion that aggregation can always combine all data into a single packet. We then build a phase space representation for arbitrary application-specific data, and build a new aggregation protocol that uses the phase space representation to significantly reduce the errors v.s. traditional aggregation protocols

Whatever the nature of the application, WSNs generate data. This would be because of the primary goals of most sensor networks: gathering and processing data about the environment around the nodes. Also, for most applications, the more data that can be gathered, the better. One motivation behind this is that if excess data is gathered, it can often be used to improve the quality of the end results from the application, and that the "cost" of gathering more data is often not thought of as a major constraint in other data-gathering systems. Additionally, with increasing numbers of nodes, the volume of data becomes ever greater. Therefore, one of the fundamental problems for these networks is managing the outputted data.

Given the energy and space limitations of WSNs, moving increasing quantities of data to a sink node/end-user computer where the data can be stored and analysed will reduce the operational lifespan of such a network. For WSNs, gathering more data than is strictly needed is no longer the safe option. It can be concluded that reducing the amount of data that needs to be gathered is a required goal in order to achieve usable network lifetimes. This goal, however, conflicts with the information gathering purpose of the networks. A network that transmits no data (for example) would be very efficient, but not very useful.

We have the difference here between data and information: applications need information, which can be seen as the end result of data once it has been processed. We do not necessarily need to transmit the data all the way to the sink node before it is processed,

Most of the content in this chapter has been published as "Foxtrot: phase space data representation for correlation-aware aggregation" by Tom Parker and Koen Langendoen at the Fourth Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2007) [4]

and so the area of in-network processing techniques [35] - of letting either individual nodes (or several nodes working together) do some initial processing of the raw data into information before it is transmitted further into the network - was formed.

One particular use case of note for in-network processing of sensor data is that there will be some parts of the raw data that are not required to get the information that the application actually needs in the end, but would normally be transmitted anyways. This idea is addressed by aggregation techniques, which allow an application to decide "this data is redundant/unimportant and can be discarded", and so reduce the network data rate while still providing a particular set of required information.

In this chapter, we look at aggregation techniques, and show what can be done to improve their design by re-examining their core abstractions.

6.1 Existing work

Much work has already been done in both aggregation, and in querying mechanisms for WSNs that indirectly use aggregation techniques (e.g. SQL-based techniques like TinyDB [78], TAG [77] and STREAM [8]) for whole network queries. Given that one of the major purposes of statistical techniques is reducing large bodies of data down to a smaller set of useful representative values, their use seems an obvious good choice for aggregation techniques.

Averaging is a popular choice [77, 78] (partly because of the Central Limit Theorem [142]), as this can reduce any number of data points down to just an averaged value plus a count of the elements merged (the count is required to aid merging of multiple averages at later nodes). This has several major advantages - it is fully distributed; the end result is always very small (and the same size) regardless of how many data points were initially available; processing and space costs are minimal; and an average value for a region is a common example query asked by many proposed sensor network applications. Overall, it is a very easy (and common) question to answer, and one that many aggregation protocols have optimised towards.

In general, there is a wide variety of possible aggregation functions, and Madden et al. [77] (based on earlier work by Gray et al. [40]) proposed a taxonomy for aggregation functions, describing four properties of interest. They were

- Duplicate sensitivity - will the aggregate results change if an input value is duplicated?
- Exemplary/Summary - does the aggregate provide particular "exemplary" values, or a summary of all values?
- Monotonicity - will any combination of two partial aggregation states always be larger than the individual values (or alternately, always smaller, but not both in one aggregation function)?

- State size type
  - Distributive - partial state size is the same as the end state size
  - Algebraic - partial state size is not the same as end state size, but is a constant size
  - Holistic - partial state size is proportional to the number of data items i.e. little to no partial aggregation can be done
  - Content-sensitive - partial state size is proportional to some (statistical) property of the data values.
  - Unique - partial state size is proportional to the number of unique data items. Can be considered a special case of content-sensitive.

Madden et al. described a generic system (TAG) that was capable of doing aggregation regardless of the properties of the aggregation function, based upon an SQL-like language that described how to do aggregation, with the addition of the notion of "epochs" to specify that aggregation would only be done on sets of values that were all gathered within the same limited time frame.

6.2 Problems

The one question that these systems fail to ask is: is the answer useful? Take the common example of "what's the average temperature on this floor?", and using the answer as part of a feedback loop to keep the temperature at 20 degrees celsius. Imagine that there are 5 sensors on a floor of a building and the answer is 22 degrees. The common assumption would be that most of the sensors have a temperature of about 22 degrees, and so the system would drop the temperature by 2 degrees. But then a report comes in of a room at 18 degrees. What actually happened is that 4 of the sensors were at 20 degrees, and the 5th sensor had been placed near the output vents of a computer which rose the temperature around that node to 30 degrees. The averaging algorithm, without any correlation awareness, merged all the values into a single value that says nothing about the actual nature of the true temperature values, and thus too much information was discarded.

There are in fact only two interesting scenarios for averaging: a) all data values are approximately similar (or at least vary around a common centre), in which case a sampling of a subset of the nodes would get as good an answer as averaging, but with less network traffic; or b) data values vary widely (as in the example), resulting in a result that bears no resemblance to the actual data.

Alternately, we could use the median of a set of values instead. However, for an accurate median we need all the original data points at a single point. Q-Digest [120] attempted to reduce this problem by only transmitting a subset of the candidate data values, and providing a method to merge candidate sets together. This gave a reasonable approximation to the median (as well as other statistical values) while reducing packet

rates, but ran into the same problem as averaging: is the answer useful? In the temperature example above, it would have given us the 20 degree value (and probably also the 30 degree value depending on the level of merging). This is an improvement over averaging, but in situations with a larger number of nodes, rare events will still be discarded in favour of lots of information about common events. Also, Q-Digest gives us no location awareness; in the temperature example, we would be unable to locate the problematic sensor without using other techniques.

Statistical techniques are mostly limited by their requirements for needing most of the data to be merged in one location, combined with the issue of merging multiple subsets that have been previously merged. Further work in this area could possibly reduce these two problems, but we would still need to find a technique that gives useful answers, and that is a much harder problem.

Unfortunately, existing statistical aggregation work is not correlation aware: all data points are automatically considered as inputs for the aggregation mechanism, without consideration to the nature of the data. Additionally, rare events (e.g. a single sensor with different data) are not considered statistically significant, whereas for WSN applications a single sensor reading may well be important, e.g. only one sensor is attached to a tree that is on fire. Discarding the bulk of unimportant data (limiting data from the large areas of a forest that are not on fire) whilst keeping the useful information should be a focus for WSNs. Location data is also vital for WSNs ("Which tree is on fire?" See also Chapter 4), and when correlation awareness is considered, we must also check where multiple data points were measured in order to determine whether they can be aggregated.

We conclude that a newer approach, focused on the usefulness of the end result to the users of the application, is required for an improved aggregation technique. We also conclude that the major goal of aggregation is a trade-off between information loss and packet data rate reduction, and that any aggregation technique that attempts to discard packets without considering what level of information loss this will cause - "blind" aggregation - is fundamentally flawed. This is a problem because of the notion that standard statistical techniques represent a "good" summary (for values of "good" for typical sensor networking applications) of a large collection of data. These summaries are an abstraction away from the true data values, but not a good abstraction for our purposes (due to the blind discarding of data), and therefore this needs rethinking.

In this chapter, we describe a new way to summarise large data sets, and describe an improved aggregation algorithm using this new method of data summary. We then proceed to compare this new method to aggregation without correlation awareness, which previously resulted in variable (often high) levels of error in the end results, and show our new method limits errors in the end results for a variety of input data scenarios.

6.3 Phase space representation

In order to consider how to create a technique that would combine correlation awareness with location knowledge, we turned to phase space representations of the data. Phase space [143] uses an abstract n-dimensional space to represent all of the possible states

of a system. Each degree of freedom in the space represents a different data value. Our initial approach to this focused on single sensor values, plus their associated location data. We later realised that the same techniques could not only be used to combine multiple sensor values, but to also incorporate other data sources.

In total, we identified three possible sources of data:

- Raw sensor values (e.g. humidity and temperature)
- Internal node data sources (e.g. location data)
- Functions of other data sources (e.g. rate of change)

The state of a node at a particular instant in time can be represented by a point in phase space defined by the values of all of the sources of data being used. Most applications would normally only be interested in a small number of sensors, plus 2 dimensions for location data, but the capability for extra data sources is automatically available. Irregardless of what data is being used, the data can still be represented only as an abstract concept of a series of values without any knowledge of which of the three categories the original data source was. Individual axes may however specify certain source specific limits on what can be done with data in that particular axis.

The basic data unit is that of a point in phase space, but we also want to be able to merge data points into larger regions also within the same phase space. A region in phase space represents a range of values. A region is defined by a set of numbers $\{min_1, ...min_n\}$ and $\{max_1, ...max_n\}$ for an n-dimensional space, and covers all points of the form $\{v_1, ...v_n\}$ such that $\forall x, x \in \mathbb{N}, 1 \leq x \leq n, min_x \leq v_x \leq max_x$. A point is defined as a zero-sized region i.e. $\forall x, x \in \mathbb{N}, 1 \leq x \leq n, min_x = v_x = max_x$

The use of phase space regions for data representation can be compared to Gray et. al's earlier work on Data Cubes [40], especially Dryeson's work on incomplete Data Cubes [30]. Phase space is however only concerned with numbers (as opposed to the options for text values used in Data Cubes), and manipulation and combination of the individual data points is therefore a valid option. If text values are included, then describing a region within the text is more difficult.

For example, the numbers 10, 11, and 12 can be described as the 1-dimensional region 10-12, whereas the equivalent for the words North, South and East is more difficult. The region for a set of text can only be correctly described by an application-specific knowledge of the semantics behind the text, but numbers can be manipulated without any knowledge about what they represent. The "10, 11, and 12" of the example could be temperature values, distances, ADC values or anything else, and the description of "10-12" still applies.

6.3.1 Region merging

Aggregation can now be specified as merging of multiple phase space regions into a different (generally smaller) quantity of phase space regions. We also need to note that some regions may not be mergeable, and that any processing time spent attempting to

merge unmergeable regions is effectively wasted. Therefore, one design aim is that if the merging fails, it should fail as early as possible to reduce wasted effort.

Figure 6.1: Greedy merging of two points

An initial greedy approach to merging would be simply to merge any and all points into a large region that contains all of them. This approach has a number of problems, as demonstrated in Figure 6.1. Specifically, the approach is too greedy, and ends up describing regions that not only contain the original points, but also large areas that are not in the original data set, and so can provide results that differ significantly from the original data. Additionally, greedy merging of sensor data will result in large ranges in the results e.g. for the temperature example in Section 6.1, we would get the range "20-30 degrees", losing significant amounts of information. We do however still want to be as greedy as we can in the merging algorithm, as a greedier algorithm will result in being able to merge greater numbers of regions into a single region. Therefore, in order to find an algorithm that is greedy enough, but not too greedy, we need to constrain how data points are merged, and also decide if some points can in fact not be merged at all. A particular set of data points are only mergeable if all sources are mergeable for the particular points.

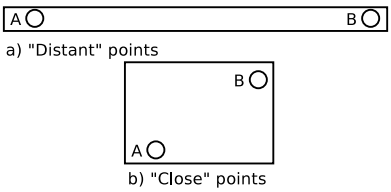


Figure 6.2: Location merging examples

The constraints required for sensor data and for location data differ in their requirements. For location data, we want to be as greedy as we can, provided that the end region does not cover areas that were not implied by the original data. For sensor data, we have more fixed constraints. For example, an application may specify that an acceptable level of data loss from a temperature sensor is 1 degree. In this case, if two points are further apart than 1 degree in the temperature dimension, then they cannot be merged. Conversely, if they are no more than 1 degree apart, they can always be merged. This contrasts with location data, where "close" values may create over-sized regions whereas "distant" points may not. See Figure 6.2 for examples of these two cases. In the "distant"

case we have nodes in all of the corners of the created region, making the assumption that the central region contains similar values a reasonable inference. In the "close" example, we have no points in the top-left and bottom-right areas, so merging these points would infer much more without evidence to back up the assumption. If, on the other hand, we had data from nodes in the top-left and bottom-right areas, then creating the "close" region would be much less likely to cause problems. Another factor that makes this form of estimate more reliable is the use of proper heuristics for dealing with overlapping regions, which we will look at in Section 6.4.3.

6.3.2 Constraints

Given the two differing forms of constraint, we define two classes of source data: statically and dynamically limited. In general, these will correspond to sensor data and location data respectively, but this may vary on a per-application basis, and for the purposes of merging we only need to know the class of a data source.

Statically limited data sources have the criteria that a data point with a particular value from this source can be merged with any other data point that is not further away (difference between two values) than a specific value e.g. a temperature source may say that the limit is 1 degree. This means that two temperatures that are more than 1 degree apart will never be merged, thus giving a guaranteed limit on the amount of information that will be discarded.

6.3.3 Dynamically limited sources merging

Dynamically limited sources are more complicated, and are merged as a set (i.e. all dynamic sources are tested at the same time). They have the advantage that they have no fixed limits as to which can be merged, but instead have a series of criteria to guarantee that the created region does not expand into regions that are not suitably covered by the original regions used. The method works with the corner points of all the regions involved, and not expanding a region in a particular direction unless there are suitable points in that direction to indicate that it is safe to expand in that direction, in order to avoid overly greedy merging.

To merge a set of regions defined by dynamically limited sources (see also Figure 6.3):

1. Define an initial zero-sized box in the centre of all the original regions, called Ψ.
2. Λ = set of all corners of the regions.
3. For each dynamically limited source α, perform steps (a) to (e) twice, firstly for the positive direction, and secondly for the negative direction. See Figure 6.4 for an example. The current direction is specified as Υ.
   (a) The set of test directions is defined as as the cartesian product $A_1 \times \ldots \times A_n$, such that $A_\beta = \{A_{pos}, A_{neg}\}$ (positive and negative) for all of the dynamic sources A_β and $\beta \neq \alpha$.

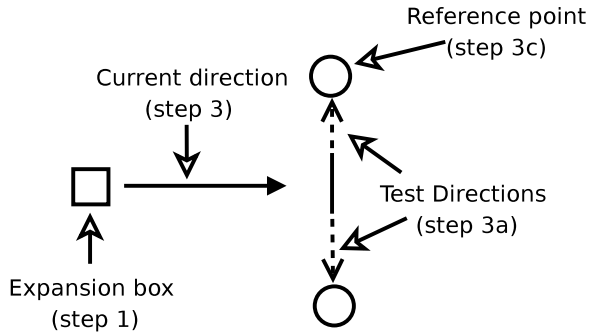


Figure 6.3: Dynamically limited sources merging example in 2-D

(b) Initialise a result variable Δ to the maximum possible value of α if Υ is positive, otherwise to the minimum possible value.

(c) For each test direction A_β, check if there exists a point in Λ that satisfies each direction in A_β for Ψ. For example, given a test direction $\{x_{pos}\}$, the point must have an x co-ordinate greater than or equal to the largest x co-ordinate of Ψ. Similarly, for $\{x_{neg}\}$, the point would need to have an x co-ordinate smaller than or equal to the smallest x co-ordinate of Ψ. If we have one or more points that satisfy this criterion; then if Υ is positive, set Δ to the minimum of all of their α values, else set Δ to the maximum of all of their α values.

(d) If we were unable to find one of the test values in step c), quit as these regions are not mergeable.

(e) If Υ is positive, set the maximum α value for Ψ to Δ, else set the minimum α value for Ψ to Δ.

4. If we have completed step 3 without quitting, then Ψ is a merged form of the original regions.

5. For each original region, check it against Ψ. If Ψ completely covers the original region, we can discard the original region. Alternately, Ψ may partially cover the original region. If Ψ completely covers the original region on every dynamically limited source aside from one, remove the part of the original region that is within Ψ. Otherwise, we cannot do anything with the original region.

6. If we were unable to completely cover any regions in step 5, then we have generated an extra region, and so the original regions were not mergeable. Otherwise, return the revised set of regions, including Ψ.

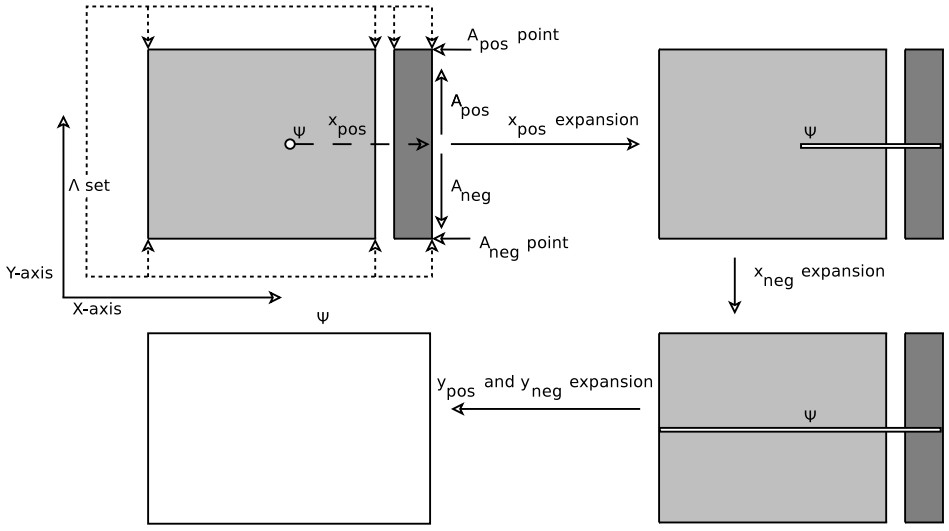


Figure 6.4: Example of Dynamically limited sources merging Step 3

6.4 Foxtrot

Foxtrot is built from the new data abstraction and merging concepts introduced in the previous section, creating a novel fully-distributed data aggregation protocol.

In common with any other aggregation protocol that wants to do in-network aggregation, Foxtrot requires a source-to-sink routing protocol that allows packets passing through a particular node to be altered and/or dropped depending on the choices of the aggregation protocol. In fact, the easiest way for aggregation protocols to do this is is for the routing protocol to not automatically forward incoming packets, but to hand them to the aggregation protocol, which then may later give (some) packets back to the routing protocol for further forwarding. One such protocol is described in Chapter 3 (Guesswork), but Foxtrot will work with other routing protocols as well.

Foxtrot is an event-triggered protocol, with the events being the timers for the periodic data input interval defined by the application, and packets arriving from other nodes. We first describe the interactions between Foxtrot and other code in the software stack, and then describe the set of actions necessary to implement Foxtrot by defining what to do when events (new data in from either the application layer or in packets from other nodes) occur.

6.4.1 Interfaces

Foxtrot requires input from both the application and other modules in the software stack, and provides output data to the application:

- Application Input
  - All nodes need to initialise Foxtrot, providing the length of the periodic "epoch" interval and the values for the static limits (Section 6.3.2).
  - Data source nodes need to provide data from their sources to Foxtrot on request, which will occur every "epoch". Notably, this is in the "raw" form of the data, as opposed to the phase-space form (which is generated internally by Foxtrot).
- Interaction with other modules
  - Foxtrot needs location data, generally from a Localisation module (or external location information hardware e.g. GPS)
  - Foxtrot needs timing functions in order to implement "epoch" intervals (which will generally require network-wide time synchronisation).
  - Packets are handed by Foxtrot to a routing protocol, which routes the packets to the next hop node in the route to the sink and then gives the packet back to Foxtrot.
- Output to Application
  - Foxtrot provides data to the sink node(s) in the merged phase-space form, providing both location and data source information.

6.4.2 Source nodes

As we noted above, data source nodes hand data to Foxtrot when requested (once every "epoch") . Foxtrot then converts the data into the phase space data representation along with the location data, and hands it off to the underlying routing protocol for forwarding towards the sink. If packets arrive at a node, then Foxtrot will attempt to merge them together along with any other packets currently stored at this node, and then hand over the results to the routing protocol.

Currently, Foxtrot does dynamically-limited sources merging for the location data, generating a series of sets of possible merged packets, which are subsets of the complete set of locally held packets that the algorithm in Section 6.3.3 considers mergeable. Each set of possible merged packets is then handed to an internal "threshold merging" function, which uses the static limit input values provided by the application to make further decisions about whether the particular set can be considered mergeable. One potential future extension to Foxtrot would be to allow applications to provide their own "merging test" function, allowing applications to make more finer grained decisions about merging e.g. variable thresholds for particular input sources depending on the current values of other sources.

6.4.3 Sink node

Sink node(s) receive packets consisting of regions in the phase space for the application. This data should then be handed over to the application (which may well then give the data to the sink-connected PC, store the data for future reference, or any other action that the sink node wishes to do). However, the format in which this data is provided to the application brings up a number of issues. Applications will probably have to be adapted to Foxtrot, but this applies to most other aggregation protocols as well (e.g. an application designed for raw data would have to be changed to use averaged values correctly). The current implementation provides the data in the standard Foxtrot format i.e. a series of phase space regions. These can be combined to provide a complete picture of the network without much effort.

One issue that can come up is that it is possible for the regions gathered by the sink node to overlap. How an application wishes to deal with this problem may well vary. The simplest options is to provide all of the possibilities for overlapping nodes e.g. a node may be marked as either being between 20 and 21 degrees celsius, or being 30-31 degrees celsius. Each measurement comes from a separate region, but because of the merging, it is unknown which answer is correct. Foxtrot does guarantee that the correct value does not lie outside the reported regions, but it is difficult to eliminate the wrong option. A number of heuristics ("pick the smallest box", "lower values are more likely", etc) have been tested against various application scenarios, but the best option will be application-specific. Alternately, the ranges can be simply averaged. This will provide less accurate values at the uncertain points, but the values will represent a reasonable compromise between the various choices. Notably, v.s. conventional averaging, Foxtrot does provide information about which nodes have uncertain values, and which are certain, which may also be of use in some applications.

6.4.4 Timing issues

The simplified protocol model detailed above does not deal with the timing issues common to all in-network aggregation protocols. The first major problem is that periodic data measurement does not in general result in synchronised data. For example, if a particular application measures data every 10 minutes, and Node A's data can be aggregated with Node B's data, we have no guarantee that the two nodes will measure data at the same time, and therefore there could be up to a 10 minute delay between the measurements from the different nodes. This means that to allow aggregation, a node will have to delay the forwarding onwards of a packet for a much longer time than if the nodes are synchronised.

In order to solve this problem, we assume the existence of a time synchronisation protocol giving us synchronised cross-network timers (e.g. the Network Time protocol from Section 2.6.2). This also allows for better results for many scientific applications, as being able to know that the sensor data represents a snapshot of the monitored area over a short period of time is generally more useful than a measurement spread over a larger period, especially for cases where the source of the data values is changing rapidly.

The second problem is how much a protocol should delay before sending a packet onwards, and this has been dealt with in some detail in earlier work [2, 63, 125]. At the moment, we are using values based on knowledge of the routing for the whole network, with a delay value based on the number of hops from the current node to the highest hop-count child node in this part of the tree. Therefore, the basic in-network node protocol described in Section 6.4.2 is expanded to note that we delay before passing any packets (both locally generated and from other nodes) onto the routing protocol, and there is therefore an additional timer event that signals when to give all currently locally stored packets to the routing protocol for sending onwards to the next hop node.

Notably, Foxtrot only requires a solution to the problem of how long to delay a packet. Our current solution to the delay problem also requires a solution to the synchronisation problem, but other solutions can also be used with Foxtrot. Inaccurate answers will result in less optimal results (because of later and/or less aggregation) than correct ones, but Foxtrot will still work.

6.5 Results

We tested Foxtrot in two ways; firstly in simulation against averaging and Q-Digest, using a generic "smart" routing protocol; and secondly as a TinyOS implementation.

Our two metrics of interest were information loss and the number of transmitted packets. Information loss was calculated as the average per-node difference between the value received by the sink node and the true data value of a sensor at each source node. To handle multiple overlapping regions with Foxtrot, we take the average of overlapped regions (see Section 6.4.3). In all cases we ran the tests 20 times, and the data here is an average of those results.

The simulations modelled a grid of nodes with temperature sensors, with a variety of different floating point values for the temperature readings. All tests were done in an area of 30m by 30m, with a 14m radio range. Radio links were assumed to be bi-directional and perfect, and the average hop count to the sink node is ˜2.36 (2.8 with 25 nodes, through to 2.26 at 100). Q-Digest was run with the temperature values placed into 0.1 degree "bins" (in order to generate the integer values required by Q-Digest from our floating point temperature data), and Foxtrot with the maximum allowed temperature merging range set to 1 degree. Our "smart" routing protocol uses shortest-hop routes, with routing overhead being ignored.

We tested four scenarios for the dispersal of the temperature values:

1. *Spread*: Near-identical values, all nodes at values between 20 and 21 degrees
2. *Sparse*: 4% (1 in 25 nodes) of the nodes at between 30 and 31 degrees, all others between 20 and 21 degrees.
3. *Division*: Nodes on the left hand side are between 20 and 21 degrees, and nodes on the right hand side are between 30 and 31, with 50% of the total number allocated to each group.
4. *Random*: All nodes with random values between 20 and 31 degrees

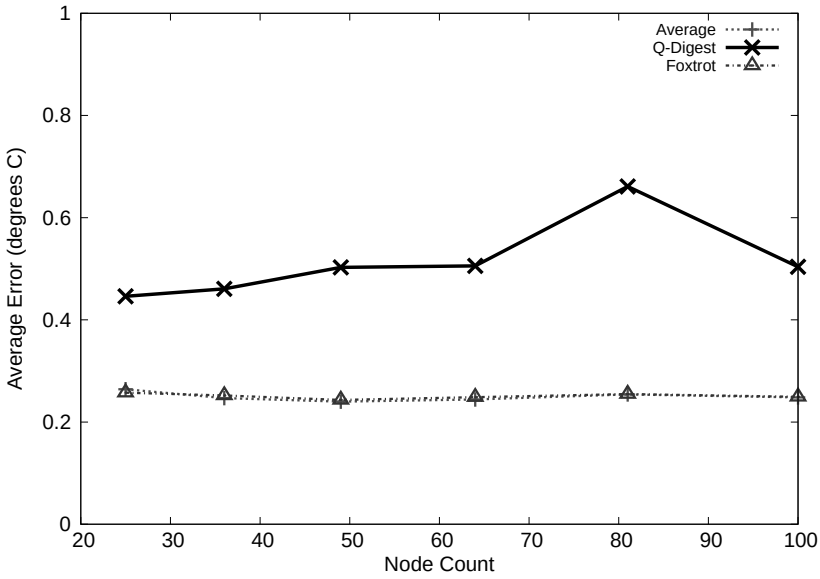


Figure 6.5: Scenario 1 (Spread)

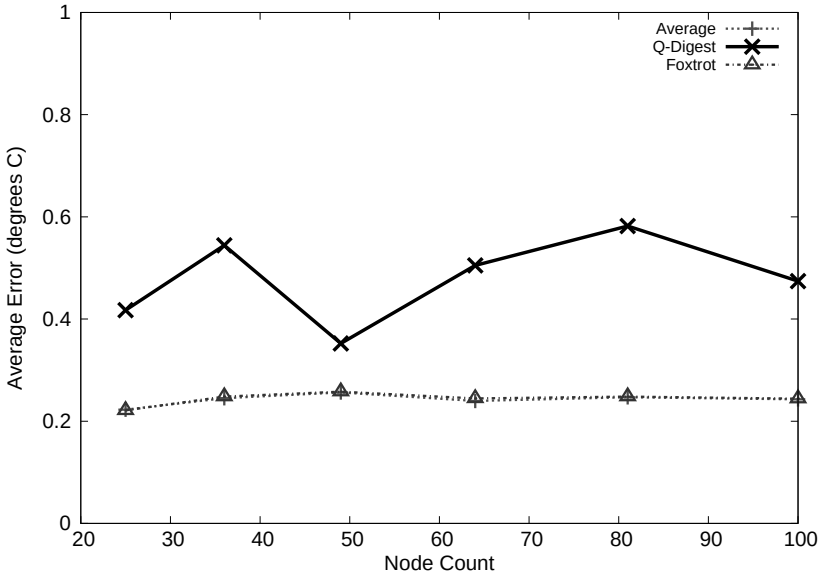


Figure 6.6: Scenario 2 (Sparse)

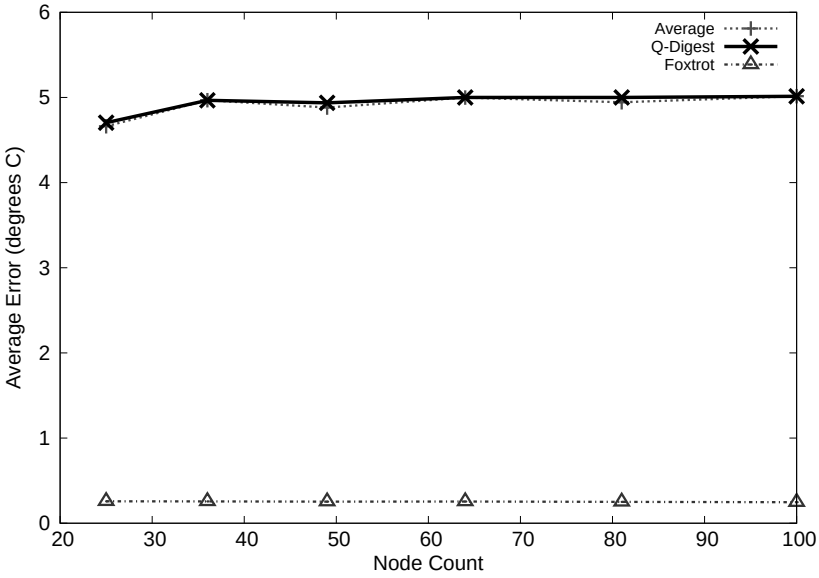


Figure 6.7: Scenario 3 (Division)

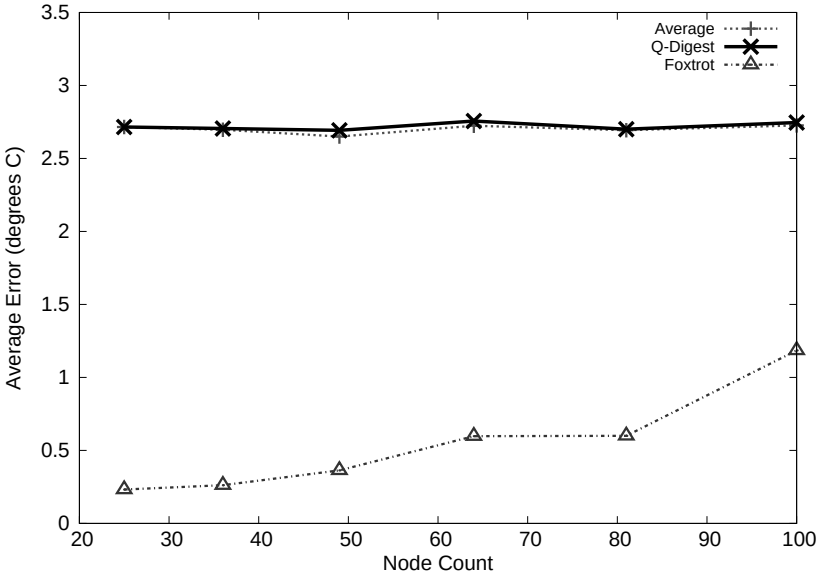


Figure 6.8: Scenario 4 (Random)

The Spread and Sparse scenarios (Figures 6.5 and 6.6), are not particularly interesting, but do show that in the situations where conventional techniques are able to achieve low error values, Foxtrot is able to achieve identical performance. Q-Digest does quite badly because it provides complete information for a series of values between 20-21 that we end up having to average to get an estimate for any given node because of the lack of location data. The averaging error of approximately 0.25 is what we would expect for this scenario, as given a mid-point of 20.5, statistically speaking for half of all the input values should be in the 20.25-20.75 range (less than 0.25 error) and half should be outside that range (in either 20-20.25 or 20.75-21, both with greater than 0.25 error).

The Division scenario (Figure 6.7) provides us with more useful results - Foxtrot's error values remain low, but the error rates for averaging and Q-Digest have both risen sharply. Similarly to Spread and Sparse, Q-Digest provides two different series of values for the overall network in this scenario, one at approximately 20 degrees, and a second at 30 degrees, reflecting an accurate picture of the network. However, due to the complete lack of location information, we are again forced to use a weighted average of the two series to derive estimates of the sensor data values, and so Q-Digest's performance is similar to averaging (˜5 degrees error). The difference here is that both averaging and Q-Digest give inaccurate results, but with Q-Digest an end user would at least be aware of the situation. With Foxtrot, we get two sets of values plus location information, allowing accurate estimates of the data values, and maintaining the low error rates shown in the first two scenarios.

Foxtrot does not perform quite as well in the Random scenario (Figure 6.8), but this is the least likely of the scenarios to actually occur, given the correlation that tends to exist within multiple nearby readings for most physical values used by sensor nodes. Despite the low likelihood of this scenario, Foxtrot is still able to get lower error values than other methods. In fact, one of the major sources of Foxtrot error is due to the overlapping issues described in Section 6.4.3, and this is also responsible for the increasing error with more nodes (which in most real world scenarios would be even less likely to have different random values). We use averaging here to resolve ambiguities, and this gives us higher errors than we might be able to achieve with more highly tuned methods.

The trade off is that Foxtrot requires more packets to be sent, as it is not necessarily able to always merge all data, which is shown in Figure 6.9. This graph shows average packets sent per node, in order so we can more easily see trends in the data. Firstly, Q-Digest and average both have exactly the same packet rate - 1 packet per node, as they merge everything. Foxtrot's packet rate varies substantially depending on the scenario, because the amount of unmergeable data in the network varies. For the Random scenario, the packet rate is fairly similar to the values for no merging, with only a reduction of 3.7%. For scenarios 1-3 (Spread, Sparse, Division), the packet rate reduction is more substantial, with an average of a 13% reduction in overall packet transmissions.

Different packet transmissions are not all the same in a real-world sensor network, and spreading packet load over the entire network as opposed to having most of the load near to the sink will also help to save power used by those transmissions due to to less contention and reduced idle listening time. Figure 6.10 shows the reduction in the number of transmitted packets v.s. non-aggregated scenarios for nodes within one hop of

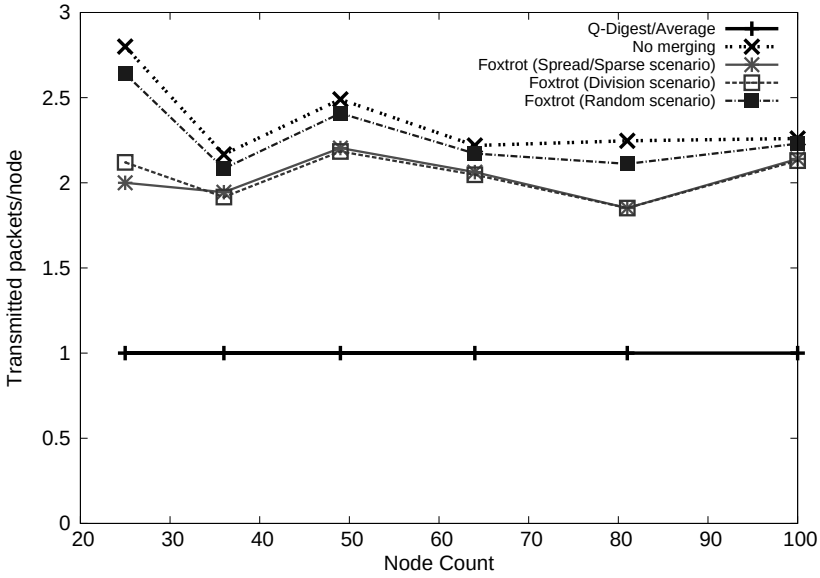


Figure 6.9: Packet counts

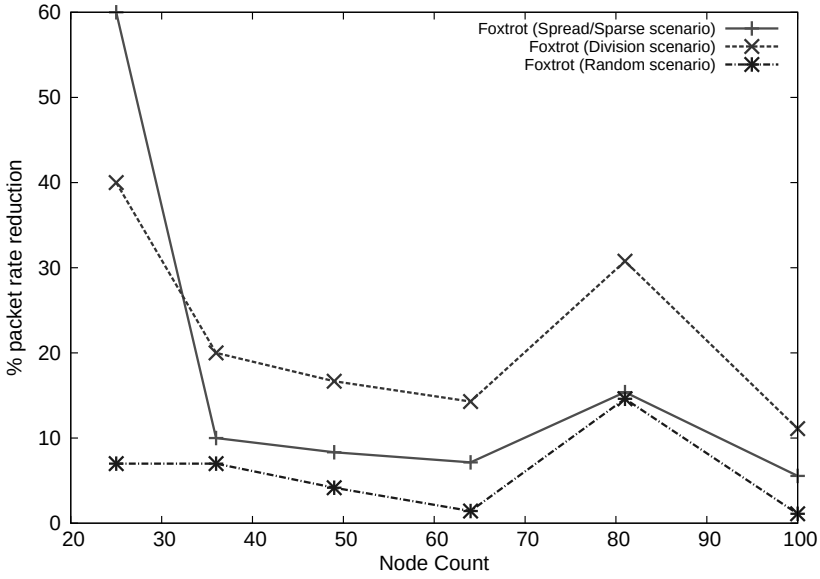


Figure 6.10: Reduction in packets for sink neighbours

the sink node. In Spread, Sparse and Division, we achieved an average reduction of 19% (with values up to 60% for some scenarios). The Random scenario had a 5.8% reduction.

To test that Foxtrot would work with actual node hardware, as opposed to just in simulation environments, we also implemented Foxtrot for TinyOS. We used Guesswork (from Chapter 3) to provide routing, but any other reliable sink-to-source routing algorithm would be a viable candidate. The resulting program for our mica2 derived nodes added up to a total of 44222 bytes of ROM. Getting exact values for the Foxtrot modules on their own is difficult, but the simple test program for the routing protocol takes up 38330 bytes in total, so a size for Foxtrot in the region of 6Kbytes is not unreasonable, and as that would be only 4.6% of the total program space of 128Kbytes, we can conclude that Foxtrot will not cause too many problems for application designers in terms of finding enough space on their nodes. Results from TOSSIM indicate that the TinyOS implementation behaves similarly to our earlier simulation data, and early testing with node hardware indicate that this still holds true for when tested on real hardware.

6.6 Sparse mapping

One of the remaining problems for Foxtrot using the phase-space representation is the difficulties in merging physical locations. We show here some early work towards a better solution, but this is still work in progress.

Sensor data tends to have a reasonable degree of correlation, and adjacent sensor nodes will tend towards similar readings, which makes merging easily viable. Physical locations tend to be less easy to merge. If we use a grid of nodes (as in Section 6.5), then each node has multiple nodes nearby that have at least one co-ordinate in common, and so merging the location data can be simple. If however the routing topology varies significantly from Manhattan routing [83] (i.e. next-hop nodes are anything other than the nodes directly north, south, east and west of the current node), then aggregating data will again be difficult. The same situation occurs in networks with non-grid deployments of nodes, e.g. the random deployments typical to most simulation scenarios.

We attempted to work around this in Section 6.3.1, and further improvements are possible. One option is allowing for a certain amount of fuzziness in the notion of "equal" values e.g. rounding all values to the nearest whole unit before considering merging. This is however flawed, because it is effectively doing greedy merging (Section 6.3.1 has more details on the problems with this), but trying to use it in only a limited way in order to allow "enough" merging but not "too much". We mention both of these terms in quotes, because the definition of "too much" and "enough" are themselves very fuzzy concepts, with the exact values being not only application dependant, but dependant on the state of the data in the network. Selection of correct values is therefore generally non-feasible.

Sparse mapping is a way to provide a better representation of the stored data than standard Foxtrot. We do this by extending the semantics for a region from "these value ranges apply across this entire region" to "these values only apply in certain parts of a region". This is done by adding a n-dimensional (where n is equal to the number of physical dimensions used, 2 in our examples) bitmap grid to the packets. Each bit

Algorithm 3 Bitmap semantics

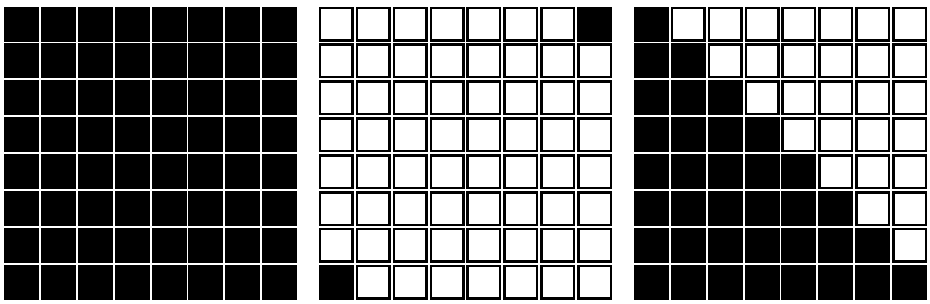
The formulas here are for a 2-D grid, but can also be expanded to 3-D (which needs more bits for a given fidelity of bitmap)

- A bitmap is B_W xB_H bits in size, describing a region from the point $(R_x, R_Y) \times (R_x + R_w, R_Y + R_H)$.
- R_W and R_H are the width and height of the region respectively.
- An individual bit covers a region b_w by b_h, where $b_w = R_W / B_w$ and $b_H = R_H / B_H$
- Each bit $b(x,y)$ covers the region $(R_x + (b_w * x), R_y + (b_h * y)) \times (R_x + (b_w * (x + 1)), R_y + (b_h * (y + 1)))$

For example, if you have an 8x8 grid for a packet with the region $(10,18) \times (10,18)$, then $R_W = 8$, $R_H = 8$, $B_W = 8$, $B_H = 8$, $b_w = 1$, $b_h = 1$ and the first bit $b(0,0)$ covers the region $(10,11) \times (10,11)$.

in the bitmap represents a subsection of the physical region specified for the packet. Algorithm 3 defines the basic semantics for a bitmap, and Algorithm 4 describes how to merge bitmaps when their respective regions are merged. Additionally, single data points start off as a completely filled grid (Figure 6.11a).

In Figure 6.11, we show a graphical representation of a series of example grids. 6.11a shows the effective grid for the old "value ranges apply across the entire region" semantics, where as 6.11b and 6.11c show two examples of original data (separate corners and diagonals) that we could not represent accurately with standard Foxtrot. Figure 6.11b in particular is a representation of a situation similar to Figure 6.1 on page 118.



(a) Full (b) Corners (c) Diagonal

Figure 6.11: Example sparse mapping bitmap visualisations

Given that we now have a better representation of data stored in a Foxtrot region, we can re-evaluate the restrictions from Section 6.3.1, which were intended to reduce information loss when only ranges were used to defined spacial information. One option

Algorithm 4 Bitmap merging

To merge two bitmaps, *A* and *B* into a new bitmap *C*, using the definitions from Algorithm 3:

1. The new bitmap *C* is initially empty, with all points set to 0.
2. For each non-zero bit in *A*, determine its covering region, *Z*
3. The corners of *Z* are within the covered regions for 4 bits of *C*. These 4 bits describe a rectangle in *C*'s bitmap, designated *T*. Some or all of these bits may be the same bit (if, for example *A* represented a single point).
4. For every bit in *T*, set it to 1.
5. Repeat steps 2-4, but for *B* rather than *A*.

would be to discard all of the spacial restrictions entirely, and some limited work has been done in exploring the consequences of this, but more work is needed in this area before it can be deployed further.

6.7 Conclusions

We have shown here that existing aggregation techniques are much more lossy than earlier estimates may have thought, and that the error rates from these protocols may vary widely over the lifetime of a network. To combat these problems, we proposed rethinking the core concept of aggregation protocols, redefining them as a limited information summary of the data in a network, focusing on the trade-off between information loss and reduced packet rates. To aid this new definition, we defined the phase space abstraction for representing data from a sensor network, incorporating the concept that not all data points can necessarily always be merged. The phase space abstraction gives us a new building block for use with data aggregation protocols.

We proposed Foxtrot, a limited information loss aggregation protocol, which uses the improved phase space abstraction. Foxtrot aggregates sensor data without significant information loss, and without discarding location information. This increase in information comes at a cost in additional packet transmissions v.s. more lossy aggregation protocols, but the resulting information is much more reliable due to consistently lower error rates.

This reconsideration of the concept of aggregation protocols points the way towards a new generation of sensor software, where application users will hopefully be willing to use aggregation techniques. Currently, many scientific application users have been cautious about the use of aggregation protocols, given the possibility of information loss. Techniques like Foxtrot, with its focus on information loss reduction within application-specific boundaries, may well help to persuade future projects to use aggregation without fearing the loss of experimental data.

6.7.1 Future Work

Foxtrot is a first generation attempt at limited information loss aggregation, and more research is required on the topic of creating aggregation protocols with similar aims to the ideas discussed in this chapter.

Foxtrot could also be expanded in a number of ways. The dynamic sources merging algorithm is relatively conservative, and further exploration of the trade-off between accuracy and greediness for merging may find better candidates. Our use of phase space regions could also be expanded to cover other polytopes, which would allow the specifying of larger regions with less of the greediness issues. The Sparse Mapping techniques in Section 6.6 are also very early work, and need more exploration before they can be integrated into deployed applications.

6.7.1.1 Routing Hints

The notion of correlation (whether multiple regions are mergeable) within Foxtrot could also be used with some routing protocols to provide additional optimisations, specifically when a locally held region is entirely enclosed by a region transmitted by another node. In this case, it is possible to discard the local region as transmitting it would not change the end results, thus further reducing required packet transmission rates.

Correlation could also be used to "hint" to the routing protocol that sending a packet via a particular node would result in packet merging (and therefore reduced overall packets needed to be sent) and so this would be a good choice for the next hop node. For example, this could be done with Guesswork (Chapter 3) by reducing the ETX values for correlated next-hop nodes discovered in the previous round of aggregation, and would create what would be effectively "data-aware clusters", but without any cluster control packet overhead at all.

6.7.1.2 Bounded-inaccuracy Foxtrot

Foxtrot (along with most other aggregation protocols) considers the raw data that it is aggregating as a series of specific values. This is however only partially true, as many sensors are at least somewhat inaccurate, and often information regarding their inaccuracies is provided in the manufacturer datasheets (e.g. $\pm 5\%$). This is generally ignored by current WSN systems. Similarly, when using a localisation system like RSL (Chapter 4), the provided location data is inaccurate, but has a limit on how inaccurate the value is (its bounding box). Given that both have inaccuracy information available, we can therefore move from using single data values to "bounded inaccuracy" values, which consist of a value and a range containing that value in which all possible other values for the data point are also contained. As Foxtrot has a combined representation for sensor data and location data, and the bounded inaccuracy values are not currently used, expanding Foxtrot to use them would be a useful future extension of the work.

Chapter 7

Conclusions

> "If you build a better mousetrap, someone will build a better mouse" - *Anon*

We set out in this thesis to re-examine the abstractions used by various different classes of typical WSN protocols; to see how the existing work in each field used abstraction and what problems were caused by the (implicit and explicit) assumptions made by those abstractions. We noted that WSNs are a hybrid field made from several predecessor fields, each with different sets of requirements and priorities to WSNs, and that most of our existing abstractions were inherited from those predecessor fields. To resolve some of these problems, we built new abstractions that better represented the real problems facing WSNs, both the problems that are WSN-specific (lack of resources, especially power) and those that we inherited (e.g. unreliability of radio links). Using the new abstractions, we then built new examples of protocols, and showed how they could improve on existing protocols - either in terms of improved performance, or by providing new capabilities utilising the improved abstractions.

In Chapter 2, we re-examined MAC protocol design, and came to the conclusion that the standard design specifications were insufficiently abstract, which resulted in MACs that were much larger than required; were unable to reuse parts of the common code that was duplicated; and limited the opportunities for expanding the designs to provide extra capabilities for higher layers, because of the necessity of rebuilding any expansion for every different MAC protocol. We instead described the λMAC framework, which showed how to build MAC protocols that could reuse much larger collections of common code, giving a substantial reduction in the amount of code required to implement a new protocol (e.g. λT-MAC was only 32% of the size of the original T-MAC implementation). This also allows MAC designers to abstract away from problems like time synchronisation, which were previously a major area of difficulty for MAC builders, but with the λMAC framework time synchronisation is done automatically.

In Chapter 3, we looked at routing protocols, and found that the standard primitive for routing protocols (single-hop reliable links) was an energy-inefficient abstraction for WSNs with variable-quality radios. We separated the problems into two groups: the

notion of *N* hops to the sink node was replaced with ETX (Expected TX cost) values, and the single-hop unicast primitive was replaced with ExOR, which better utilises the reality of WSNs as broadcast-based systems. We generalised ExOR to allow for new metrics, as opposed to its original usage of shortest-path hop counts, and delaying metric decisions. From generalised ExOR we created ExOR-ETX using ETX values and ExOR-Bcast for reliable broadcast/flooding of values. We then used these new actions to build Guesswork, an adaptive routing protocol that could work efficiently with a wide variety of link qualities. Guesswork was able to use the delayed metric decisions of generalised ExOR to delay the choice of best next-hop nodes until after packets have reached multiple potential next-hop nodes, which improved reliability with unreliable links.

In Chapter 4, we looked at localisation protocols, and discovered that the building block used by most localisation protocols (unreliable single distance values between pairs of nodes) discarded too much information about the actual underlying statistics of the sensors that provided the information, and was therefore a flawed abstraction. We instead moved to the use of probability maps to provide an improved abstraction mapping between incoming sensor values and the distance between nodes. Using the probability maps, we then built RSL (Refined Statistic-based Localisation), which also used bounding box information based on sanity boundaries for node locations. We also introduced limited broadcast of generated "pseudo" anchors to solve the problems of high processing time and dense anchor formations required by earlier work in statistic-based localisation. We also explored the standard notion of anchor nodes, showed that for most proposed application scenarios that a mobile anchor node could be implemented for similar costs to a single anchor, and that mobile anchors could be used to generate many "virtual" anchors along a path. We showed data for RSL working with mobile anchors, providing good localisation with the new abstractions.

In Chapter 5, we looked at motion detection. Normally, motion detection would either be done with specialised hardware or with the aid of localisation techniques. Given the probable absence of the former, and that we had redesigned the abstractions for localisation, we explored whether these abstractions could be used to do motion detection. We looked first at working with the probability maps, rethought the localisation abstractions further in order to use them to build force equations for springs in a mass-spring model, and used the mass-spring model to do motion detection for limited amounts of motion. We then showed that the sanity constraints of the location bounding boxes could be exploited to detect larger quantities of motion.

In Chapter 6, we looked at aggregation techniques. We found that the standard problem approached by most aggregation protocols is discovering better ways to distribute various statistical functions (e.g. average, median, count of values exceeding a threshold), and providing ways to manipulate the information for different application scenarios. We discovered that considering standard statistical functions as a usable abstract summary of data from a set of nodes did not match with the requirements of most sensor networks, as most statistical techniques treat rare values as anomalies to be discarded, as opposed to most WSN applications, where the anomalous values are the most important (e.g. "which tree is on fire?"). We instead suggested the notion of aggregation protocols as a "limited information summary", and that "blind aggregation" of data points

without considering the values was flawed. We proposed a phase-space abstraction for sensor data, showed how to merge data sets using this form, and noted when certain sets should not be merged in order to preserve the different data values. This new abstraction also preserved location data, which was discarded by all previous WSN aggregation protocols. We then described Foxtrot, which uses the phase-space techniques to build a distributed aggregation protocol.

We believe that our study of these five different areas, showing issues across a wide selection of the traditional design problems for sensor networks, is convincing evidence that there were problems with the existing abstractions for WSNs, and that re-examining the abstraction stack for WSNs results in improved abstactions and provides new ways to think about the problems facing future research. We devised new "building block" abstraction concepts for use with protocols in four of the areas (Generalised ExOR; probability maps; spring models for distance; phase space data representations), showed how the old building blocks took away too much information (i.e. were too abstract), and gave examples of how the new building blocks could be used to build new protocols. In Chapter 2 we also showed that sometimes the problem can be that systems are insufficiently abstract - demonstrating that abstraction problems can cut both ways, and that protocol designs can be both insufficiently abstract and too abstract.

7.1 Usefulness of sensor networks

One hope for this work is that the rethinking of the abstractions used by WSNs will allow for further developments of the field as a distinct research area. Recently, there has been some level of concern [45] that the eventual goals for WSNs of millions of very cheap "disposable" nodes will not be feasible, citing minimum cost limits for core components (especially sensors and batteries) even in quantities of tens of millions. If we take as a given that we will be unable to create sufficiently low cost hardware to achieve the end goals, then the resource restrictions that make this area distinct start to become less important. If a node should be disposable then a 20 cent reduction in per-unit costs by halving processor speed is vital, where as if we are paying 10 euros or more per node then saving 20 cents is less important, and so therefore we may be able to use much more resources than we currently assume.

An important question appears from the realisation that our costs are greater than we expected: what do we gain by trying to work with such resource-restricted systems, if the economic benefits of resource restriction are no longer so great? One answer to that is the work presented in this thesis, as the rethinking of the abstraction stack would be less likely to be considered in resource-rich systems, especially given that a number of the abstractions that we have taken apart (unicast inter-node links especially) come directly from the resource-rich predecessors to WSNs, and so if they caused problems in those fields they would have most likely been examined in those contexts.

There has been some work in resource-rich contexts related to the work here e.g. ExOR [13], which we expanded in Chapter 3, was originally built for 802.11 networks, aiming for higher bandwidth rather than energy efficiency, but in general in a resource-

rich context the problems due to inaccurate abstractions are not as obvious, and can often be ignored entirely. This ignorance comes at a cost in resources (processing time, power), but as they are available in greater abundance, this is generally unnoticed.

The improvements we make here, despite the possibly limited feasibility of resource-scarce WSNs in the long run, can still provide better results for resource-rich systems. A reduction in resource costs, regardless of the relative magnitude of the potential gains, is always useful. Additionally, giving new insights into the way we think about protocol design will help to provide new guides to future work and to entirely new areas of research.

7.2 Further work

The work presented here is the first generation of protocols looking at the abstraction stack in a new light, but they are only the beginning of what can be done. Further work is needed both to discover better uses for the new abstractions, and to find further abstraction problems in the design space. The new abstractions proposed here may well be proved in the future to be themselves flawed, and only a process of continual re-examination of our assumptions can allow us to continue to achieve better results in the future.

We show here a few additional common abstractions within WSNs that still need further work.

7.2.1 Layers

Indeed, the layered description that we have used to separate our work within this thesis is an example of an abstraction that is often forgotten about. Cross-layer protocols address this a little (e.g. LEACH [46] and D-MAC [75] are both MAC protocols that build routing information), but there is still the problem of what services a particular layer should provide. Our MAC work here provides a Network Time mechanism (Section 2.6.2) for use by other layers, but this is not common to most MAC protocols. Also, new sending primitives (ExOR [13], Section 3.3.2; Anycast [93]) are not supported by the "standard" MAC abstraction, so how can users build portable systems on top of them? This also occurs in other layers (does a routing protocol support an optimised "flooding" mechanism? What additional information does a localisation protocol provide beyond single point locations?) and so some work in improving the notion of a layer is required.

One option here is to move from fixed abstractions, to the notion of capabilities. For example, a MAC layer could say that it provides Broadcast, Unicast and Anycast; a Routing layer may specify source-to-sink routing for up to 5 sinks, plus 2-hop neighbourhood communication. This allows both for explicit expansion of the abstractions, and for choosing appropriate other layers that have particular capabilities. Additionally, metric attributes ("this layer requires *n*KB of memory", "O(n) messages setup overhead", etc) may also be possible, and allow for further improvements in the choices depending on the application requirements.

How to specify these capabilities in a way that is both correct and useful is an open question, as the more general issue of interface semantics is a long-running problem within software engineering. It may be possible to partially solve this problem by limiting the types of a module to the standard layer types, and only allowing a small set of attributes to each different type of module, but this may well limit the usefulness of the approach significantly.

7.2.2 Fuzzy neighbours

One of the conclusions that falls out of discarding the abstraction of reliable links is that another core concept to many sensor network protocol layers - neighbour nodes - starts to fall apart. Previously, a neighbour was considered as a binary relation - either two nodes could communicate, or they could not. With unreliable links, a wider range of possibilities for the concept of "neighbours" exists, as not only do we have communication links where only a subset of messages get through, but also asymmetric links, where the reliability may vary widely depending on which node initiates a transmission.

This leads to the question: under what circumstances is a node considered a neighbour node? If a message has been seen "recently"? If it responds to all queries? The correct answer to this will often vary in between different layers, and may well even depend on the local topology around a node, as in situations where a node thinks it has few neighbours it may wish to be more lenient with its decisions regarding which are considered neighbours. Also, not all neighbours are equal, and protocols will need to be aware of this when picking a "random neighbour" if they wish to have an efficient implementation. Random neighbours are also difficult to select when the node density is high (i.e. lots of potential neighbours), as only a subset will be capable of being stored in a finite amount of memory, and deciding when to discard one neighbour in favour of another may well also be protocol dependant.

One direction in which this could be taken is the notion of "fuzzy sets", where each member of the set has a probability value associated with it specifying the "grade of membership" of an item, and this could potentially work well with the concept of unreliable links.

7.2.3 Topology randomisation

Most simulation environments specify a certain number of nodes randomly scattered within a region of a specified size. One problem with this formation is that if the distribution is sufficiently random (which given that all of the locations are generally being created by the same pseudo-random number generator is a reasonable assumption, and is in fact the basis for why "random" locations are chosen) is that the node density across the area will be very even, especially as the number of nodes increases for a given size of area. This is a problem for evaluating many WSN algorithms as their locality dependence (discussed initially in Section 1.2.3, and further in the chapters on individual protocols) means that the performance of a protocol is dependant on the density of nodes. This lack of variety in the node density means that a protocol is only being tested at a small range

of node densities, and although the standard approach of varying the number of nodes in different experiments will test with different densities, in each scenario only a small range in node density is present.

Moving to scenarios with a variety of node density variations i.e. not only uniform, but other potential variants (random across a wide range, linear change in a particular direction, etc), would help to create more trustworthy simulations. One of the major problems for simulation environments is their level of accuracy v.s. real deployments, and this is normally a trade-off between processing time and fidelity. Doing node density variations would not require more processing time per simulation (although more simulation runs may be required), merely a little more effort at topology generation time, and the improvement in the trustability of the simulation data may well be significant.

Bibliography

[1] Sun SPOT system: Turning vision into reality. White Paper, June 2005.

[2] T. Abdelzaher, T. He, and J. Stankovic. Feedback control of data aggregation in sensor networks. In *Conference on Decision and Control*, 2004.

[3] K. Akkaya and M. Younis. A survey on routing protocols for wireless sensor networks. *Ad Hoc Networks*, 3(3):325–349, 2005.

[4] J. Al-Karaki and A. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(6):6–28, 2004.

[5] M. Ali, T. Suleman, and Z. A. Uzmi. MMAC: A mobility-adaptive, collision-free mac protocol for wireless sensor networks. In *Proc. 24th IEEE Performance, Computing, and Communications Conference (IPCCC'05)*, pages 401–407, Phoenix, Arizona, USA, April 2005.

[6] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, et al. A line in the sand:a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.

[7] *Atmel*©. AVR 8-Bit RISC - Datasheets, 2005.

[8] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.

[9] B. Bakshi, P. Krishna, N. Vaidya, and D. Pradhan. Improving performance of TCP over wireless networks. *17th International Conference on Distributed Computing Systems (ICDCS), May*, 1997.

[10] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[11] L. Balzano and R. Nowak. Blind calibration of sensor networks. *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 79–88, 2007.

[12] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2, 2001.

[13] S. Biswas and R. Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):69–74, 2004.

[14] D. M. Blough and P. Santi. Investigating upper bounds on network lifetime extension for cell-based energy conservation techniques in stationary ad hoc networks. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 183–192, New York, NY, USA, 2002. ACM Press.

[15] B. W. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[16] D. Braginsky and D. Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM Press.

[17] N. Bulusu, J. Heidemann, V. Bychkovskiy, and D. Estrin. Density-adaptive beacon placement algorithms for localization in ad hoc wireless networks. In *IEEE Infocom 2002*, New York, NY, June 2002.

[18] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, Jan-Mar 2004.

[19] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad-hoc networks. *Cluster Computing*, 5(2):157–167, Apr. 2002.

[20] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri. Monitoring civil structures with a wireless sensor network. *IEEE Internet Computing*, 10(2):26–34, 2006.

[21] Chipcon AS. CC1000 Datasheet (rev. 2.3), 2005.

[22] V. Claesson, H. Lonn, and N. Suri. Efficient TDMA synchronization for distributed embedded systems. *Reliable Distributed Systems, 2001. Proceedings. 20th IEEE Symposium on*, pages 198–201, 2001.

[23] S. Coleri-Ergen and P. Varaiya. PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. *IEEE Trans. on Mobile Computing*, 5(7):920–930, July 2006.

[24] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pages 171–180, Los Angeles, CA, USA, Nov. 2003.

[25] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[26] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, New Jersey, October 2002. ACM SIGCOMM.

[27] M. Ditzel and F. Elferink. Low-power radar for wireless sensor networks. *Radar Conference, 2006. 3rd European*, pages 139–141, 2006.

[28] L. Doherty, K. Pister, and L. E. Ghaoui. Convex position estimation in wireless sensor networks. In *IEEE Infocom 2001*, pages 1655–1663, Anchorage, AK, Apr. 2001.

[29] A. Dunkels, T. Voigt, and J. Alonso. Making TCP/IP Viable for Wireless Sensor Networks. In *Proc. of the First European Workshop on Wireless Sensor Networks (EWSN'04)*, Berlin, Germany, Jan. 2004.

[30] C. E. Dyreson. Information retrieval from an incomplete data cube. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 532–543. Morgan Kaufmann, 1996.

[31] C. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica. A Modular Network Layer for Sensornets. *Proceedings of the ACM Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[32] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *IEEE International Conference on Communications (ICC)*, New York, Apr. 2002.

[33] A. El-Hoiydi and J.-D. Decotignie. WiseMAC: An ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks. In *Proc. of the Ninth Int. Symp. on Computers and Communications, 2004 (ISCC 2004)*, volume 1, pages 244–251, July 2004.

[34] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: An operating system architecture for application-level resource management. In *Symposium on Operating Systems Principles*, pages 251–266, 1995.

[35] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *5th ACM/IEEE Int. Conf. on Mobile Computing and Networks (MobiCom '99)*, pages 263–270, Seattle, WA, Aug. 1999.

[36] A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem. Distributed online localization in sensor networks using a moving target. In *Proceedings of the third international symposium on Information processing in sensor networks*, pages 61–70. ACM Press, 2004.

[37] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *ACM Conf. on Programming Language Design and Implementation (PLDI)*, pages 1–11, San Diego, CA, June 2003.

[38] M. Gerla, T. Kwon, and G. Pei. On demand routing in large ad hoc wireless networks with passive clustering. In *Proceedings of IEEE WCNC 2000*, Sept. 2000.

[39] D. Goense, J. Thelen, and K. Langendoen. Wireless sensor networks for precise Phytophthora decision support. In *5th European Conference on Precision Agriculture (5ECPA)*, Uppsala, Sweden, June 2005.

[40] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[41] I. Gupta, D. Riordan, and S. Sampalli. Cluster-head election using fuzzy logic for wireless sensor networks. *Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual*, pages 255–260, 2005.

[42] G. Halkes and K. Langendoen. Crankshaft: An Energy-Efficient MAC-Protocol For Dense Wireless Sensor Networks. *EWSN 2007: European conference on Wireless Sensor Networks*, Jan. 2007.

[43] M. Handy, M. Haase, and D. Timmermann. Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In *Proc. IEEE MWCN 2002*, Stockholm, 2002.

[44] S. Hedetniemi and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. In *Networks 18*, pages 319–349, 1998.

[45] J. Heidemann. Sensornets: the Next Big Thing (broadening the definition of sensornet research). In *4th European Conference on Wireless Sensor Networks (EWSN 2007)*, Jan. 2007.

[46] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 01(4):660–670, Oct. 2002.

[47] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, Aug. 2001.

[48] J. Hill and D. Culler. Mica: a wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, Nov. 2002.

[49] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGARCH Comput. Archit. News*, 28(5):93–104, 2000.

[50] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks. In *1st Int. Workshop on Networked Sensing Systems (INSS 2004)*, Tokyo, Japan, June 2004.

[51] A. Howard, M. J. Matarić, and G. S. Sukhatme. Relaxation on a mesh: a formalism for generalized localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1055 – 1060, Wailea, Hawaii, Oct 2001.

[52] IEEE standard 802.11. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications for 802.11a and 802.11b, 1999.

[53] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 457–458, 2002.

[54] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobile Computing and Networking*, pages 56–67, 2000.

[55] K. Jamieson, H. Balakrishnan, and Y. Tay. Sift: A MAC protocol for event-driven wireless sensor networks. In *3rd European Workshop on Wireless Sensor Networks (EWSN'06)*, pages 260–275, Zurich, Switzerland, Feb. 2006.

[56] D. B. Johnson, D. A. Maltz, and J. Broch. DSR: the dynamic source routing protocol for multihop wireless ad hoc networks. *Ad hoc networking*, pages 139–172, 2001.

[57] G. J. Johnson. Of metaphor and the difficulty of computer discourse. *Commun. ACM*, 37(12):97–102, 1994.

[58] J. Kahn, R. Katz, and K. Pister. Next Century Challenges: Mobile Networking for ''Smart Dust". In *5th ACM/IEEE Int. Conf. on Mobile Computing and Networks (MobiCom '99)*, pages 271–278, Seatle, WA, Aug. 1999.

[59] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Mobile Computing and Networking*, pages 243–254, 2000.

[60] V. A. Kaseva, M. J. Kohvakka, M. Kuorilehto, M. Hannikainen, and T. D. Hamalainen. RF-Based Indoor Localization for Wireless Sensor Networks. Currently in submission to "Cooperative Localization in Wireless Ad Hoc and Sensor Networks".

[61] T. Kijewski-Correa, M. Haenggi, and P. Antsaklis. Wireless Sensor Networks for Structural Health Monitoring: A Multi-Scale Approach. In *Proceedings of the 17th Analysis and Computation Specialty Conference*. ASCE, 2006.

[62] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[63] B. Krishnamachari, D. Estrin, and S. B. Wicker. The impact of data aggregation in wireless sensor networks. In *ICDCSW '02: 22nd International Conference on Distributed Computing Systems*, pages 575–578, Washington, DC, USA, 2002. IEEE Computer Society.

[64] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: Experiences from a semiconductor plant and the north sea. In *3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2005)*, pages 64–75, San Diego, CA, 2005.

[65] J. Krumm and E. Horvitz. LOCADIO: Inferring Motion and Location from Wi-Fi Signal Strengths. *mobiquitous*, 00:4–13, 2004.

[66] J. Kulik, W. Heinzelman, and H. Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *5th ACM MOBICOM*, Seattle, WA, August 1999.

[67] S. Kumar, A. Arora, and T. Lai. On the lifetime analysis of always-on wireless sensor network applications. *Mobile Adhoc and Sensor Systems Conference, 2005. IEEE International Conference on*, pages 186–188, 2005.

[68] G. Lakoff. *Contemporary Theory of Metaphor*, volume Metaphor and Thought. Cambridge University Press, 2nd edition, 1992.

[69] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[70] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Rhodes, Greece, Apr. 2006.

[71] K. Langendoen and G. Halkes. Energy-efficient medium access control. In R. Zurawski, editor, *Embedded Systems Handbook*, pages 34.1 – 34.29. CRC press, 2005.

[72] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. *Computer Networks*, 43(4):500–518, 2003.

[73] Y. Li, W. Ye, and J. Heidemann. Energy and latency control in low duty cycle MAC protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, New Orleans, LA, USA, March 2005.

[74] K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor networks for emergency response: challenges and opportunities. *Pervasive Computing, IEEE*, 3(4):16 – 23, 2004.

[75] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks. In *Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, Santa Fe, NM, Apr. 2004.

[76] G. MacGougan, G. Lachapelle, R. Klukas, K. Siu, L. Garin, J. Shewfelt, and G. Cox. Performance analysis of a stand-alone high-sensitivity receiver. *GPS Solutions*, 6(3):179–195, 2002.

[77] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[78] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[79] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *First ACM Int. Workshop on Wireless Sensor Networks and Application (WSNA)*, pages 88–97, Atlanta, GA, Sept. 2002.

[80] M. S. Malone. Moore's Second Law. *Wired*, Apr. 2004.

[81] J. Martyna. Fuzzy Reinforcement Learning for Routing in Wireless Sensor Networks. In *Computational Intelligence, Theory and Applications*, pages 637–645. Springer, 2006.

[82] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297, 2001.

[83] N. Maxemchuk. Routing in the Manhattan Street Network. *IEEE Transactions on Communications*, 35(5):503–512, 1987.

[84] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, Mar. 1992.

[85] K. Muthukrishnan, M. Lijding, N. Meratnia, and P. Havinga. Sensing motion using spectral and spatial analysis of WLAN RSSI. In *2nd European Conference on Smart Sensing and Context (EuroSSC 2007)*, Oct. 2007.

[86] D. Niculescu and B. Nath. Ad-hoc positioning system. In *IEEE GlobeCom*, pages 2926–2931, Nov. 2001.

[87] D. Niculescu and B. Nath. Trajectory based forwarding and its applications. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 260–272, New York, NY, USA, 2003. ACM Press.

[88] L. Ohno-Machado et al. SMART: Scalable Medical Alert and Response Technology. http://smart.csail.mit.edu/.

[89] T. Parker and K. Langendoen. Refined statistic-based localisation for ad-hoc sensor networks. In *IEEE Workshop on Wireless Ad Hoc and Sensor Networks (associated with Globecom 2004)*, Dallas, TX, Nov. 2004.

[90] T. Parker and K. Langendoen. Guesswork: Robust routing in an uncertain world. In *2nd IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2005)*, Washington, DC, Nov. 2005.

[91] T. Parker and K. Langendoen. Adumbrate: Motion detection with unreliable range data. In *4th Int. Workshop on Networked Sensing Systems (INSS 2007)*, pages 221–228, Braunschweig, Germany, June 2007.

[92] T. Parker and K. Langendoen. Foxtrot: phase space data representation for correlation-aware aggregation. In *Fourth IEEE Conf. on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON)*, San Diego, CA, June 2007.

[93] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546 (Informational), Nov. 1993.

[94] P. Pathirana, N. Bulusu, A. Savkin, and S. Jha. Node localization using mobile robots in delay-tolerant sensor networks. *IEEE Transactions on Mobile Computing*, 4(3):285–296, 2005.

[95] G. Pei and C. Chien. Low power TDMA in large wireless sensor networks. In *Military Communications Conference (MILCOM 2001)*, volume 1, pages 347–351, Vienna, VA, Oct. 2001.

[96] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF Network Working Group, July 2003.

[97] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[98] C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Mobile Computing Systems and Applications, WMCSA '99, February 25-26, 1999, New Orleans, Lousiana, USA*, pages 90–100. IEEE, IEEE, February 1999.

[99] J. Polastre and D. Culler. B-MAC: An adaptive CSMA layer for low-power operation. Technical Report cs294-f03/bmac, UC Berkeley, Dec. 2003.

[100] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. Unifying link abstraction for wireless sensor networks. In *3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, CA, Nov. 2005.

[101] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFC 3168.

[102] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-Free Distributed Localization in Sensor Networks. Technical Report #892, MIT Laboratory for Computer Science, Apr. 2003.

[103] L. Qiu, Y. R. Yang, Y. Zhang, and H. Xie. On Self Adaptive Routing in Dynamic Environments - An Evaluation and Design Using a Simple, Probabilistic Scheme. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP '04)*, pages 12–23. IEEE Computer Society, 2004.

[104] V. Rajendran, K. Obraczka, and J. Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. In *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pages 181–192, Los Angeles, CA, Nov. 2003.

[105] T. S. Rappaport. *Wireless Communications, Principles and Practice*. Prentice Hall, 1996.

[106] R. Rashid, D. Julin, D. Orr, R. Sanzi, R. Baron, A. Forin, D. Golub, and M. B. Jones. Mach: a system software kernel. In *Proceedings of the 1989 IEEE International Conference, COMPCON*, pages 176–178, San Francisco, CA, USA, 1989. IEEE Comput. Soc. Press.

[107] M. J. Reddy. *The Conduit Metaphor*, volume Metaphor and Thought. Cambridge University Press, 1st edition, 1979.

[108] N. Reijers, G. Halkes, and K. Langendoen. Link layer measurements in sensor networks. In *1st IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2004)*, Fort Lauderdale, FL, Oct. 2004.

[109] J. Reimer. The power of Sun in a big Blackbox. *Ars Technica*, 3 Apr. 2007.

[110] RFM. *TR1001 868.35 MHz Hybrid Tranceiver*.

[111] M. Ringwald and K. Römer. BitMAC: a deterministic, collision-free, and robust MAC protocol for sensor networks. In *Proc. IEEE European Workshop on Wireless Sensor Networks (EWSN) 2005*, pages 57–69, Istanbul, Turkey, Jan. 2005.

[112] L. G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, 1975.

[113] S. Roumeliotis and G. Bekey. Collective localization: a distributed Kalman filter approach tolocalization of groups of mobile robots. *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, 3, 2000.

[114] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 11:329–350, 2001.

[115] C. Savarese, K. Langendoen, and J. Rabaey. Robust positioning algorithms for distributed ad-hoc wireless sensor networks. In *USENIX technical annual conference*, pages 317–328, Monterey, CA, June 2002.

[116] A. Savvides, H. Park, and M. Srivastava. The Bits and Flops of the n-hop Multilateration Primitive for Node Localization Problems. In *First ACM Int. Workshop on Wireless Sensor Networks and Application (WSNA)*, pages 112–121, Atlanta, GA, Sept. 2002.

[117] E. Sazonov, K. Janoyan, and R. Jha. Wireless intelligent sensor network for autonomous structural health monitoring. *Smart Structures/NDE 2004*, 2004.

[118] Y. Shang and W. Ruml. Improved MDS-based localization. *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 4, 2004.

[119] V. Shnayder, B. Chen, K. Lorincz, T. R. F. Fulford-Jones, and M. Welsh. Sensor networks for medical care. Technical Report TR-08-05, Harvard University, April 2005.

[120] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, New York, NY, USA, 2004. ACM Press.

[121] M. Sichitiu and V. Ramadurai. Localization of Wireless Sensor Networks with a Mobile Beacon. Technical Report TR-03/06, Center for Advances Computing and Communications (CACC), Raleigh, NC, July 2003.

[122] S. Simic and S. Sastry. Distributed localization in wireless ad hoc networks. Technical Report UCB/ERL M02/26, UC Berkeley, 2002.

[123] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *2nd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2004)*, pages 1–12, Baltimore, MD, Nov. 2004.

[124] J. Sobrinho and A. Krishnakumar. Real-time traffic over the IEEE 802.11 medium access control layer. *Bell Labs Technical Journal*, 10:173, 1996.

[125] I. Solis and K. Obraczka. The impact of timing in data aggregation for sensor networks. In *In Proc. of the IEEE International Conference on Communications (ICC), 2004*, 2004.

[126] J. Spolsky. The Law of Leaky Abstractions, 11 Nov. 2002. http://www.joelonsoftware.com/articles/LeakyAbstractions.html.

[127] K.-F. Ssu, C.-H. Ou, and H. C. Jiau. Localization with mobile anchor points in wireless sensor networks. *Vehicular Technology, IEEE Transactions on*, 54(3):1187–1197, 2005.

[128] L. Stein. Challenging the Computational Metaphor: Implications for How We Think. *Cybernetics and Systems*, 30(6):473–507, Aug. 1999.

[129] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[130] G. Sun and W. Guo. Comparison of distributed localization algorithms for sensor network with a mobile beacon. *Networking, Sensing and Control, 2004 IEEE International Conference on*, 1, 2004.

[131] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. EWSN 2004*, Berlin, Germany, Jan. 2004.

[132] M. A. Taleghan, A. Taherkordi, and M. Sharifi. Quality of Service Support in Distributed Sink-Based Wireless Sensor Networks. In *2nd IEEE International Conference on Information and Communication Technologies: from Theory to Applications (ICTTA '06)*, Apr. 2006.

[133] Texas Instruments. *MSP430x1xx Family User's Guide*. SLAU049B.

[134] J. Thelen, D. Goense, and K. Langendoen. Radio wave propagation in potato fields. In *First workshop on Wireless Network Measurements (co-located with WiOpt 2005)*, Riva del Garda, Italy, Apr. 2005.

[135] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence*, 128(1–2):99–141, May 2001.

[136] F. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part II–the Hidden Terminal Problem in Carrier Sense Multiple-Access and the Busy-Tone Solution. *IEEE Transactions on Communications*, COM-23(12):1417–1433, Dec. 1975.

[137] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *3rd ACM Conf. on Embedded Networked Sensor Systems (SenSys 2005)*, pages 51–63, San Diego, CA, 2005.

[138] A. Varga. The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.

[139] E. W. Weisstein. Circle-Circle Intersection. http://mathworld.wolfram.com/Circle-CircleIntersection.html. From MathWorld - A Wolfram Web Resource.

[140] D. A. Wheeler. SLOCCount. http://www.dwheeler.com/sloc/, 2001.

[141] K. Whitehouse and D. Culler. Callibration as parameter estimation in sensor networks. In *First ACM Int. Workshop on Wireless Sensor Networks and Application (WSNA)*, pages 59–67, Atlanta, GA, Sept. 2002.

[142] Wikipedia. Central limit theorem — Wikipedia, The Free Encyclopedia, 2006. [Online; accessed 6-December-2006].

[143] Wikipedia. Phase space — Wikipedia, The Free Encyclopedia, 2006. [Online; accessed 6 December 2006].

[144] Wikipedia. On-off keying — wikipedia, the free encyclopedia, 2007. [Online; accessed 19 September 2007].

[145] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pages 14–27, Los Angeles, CA, Nov. 2003.

[146] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52–58, 2001.

[147] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *21st Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1567–1576, New York, NY, June 2002.

[148] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated, adaptive sleeping for wireless sensor networks. *ACM/IEEE Transactions on Networking*, 12(3):493–506, June 2004. A preprint of this paper was available as ISI-TR-2003-567.

[149] O. Younis and S. Fahmy. HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 03(4):366–379, Oct. 2004.

[150] L. A. Zadeh. *Fuzzy Sets*, volume 8 of *Information and Control*, pages 338 – 353. 1965.

[151] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[152] H. Zhang and J. Hou. On deriving the upper bound of α-lifetime for large sensor networks. In *MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pages 121–132, New York, NY, USA, 2004. ACM Press.

[153] J. Zhao and R. Govindan. Understanding Packet Delivery Performance In Dense Wireless Sensor Networks. In *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, pages 1–13, Los Angeles, CA, Nov. 2003.

[154] T. Zheng, S. Radhakrishnan, and V. Sarangan. PMAC: an adaptive energy-efficient MAC protocol for wireless sensor networks. In *19th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS 2005)*, pages 65–72, Denver, CO, Apr. 2005.

[155] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 125–138. ACM Press, 2004.

Summary

Sensor networks is a field descended from a number of other fields, primarily distributed systems, but then with additional restrictions and problems (power limits; physical locality issues) and new design ideas (source-to-sink routing; data aggregation). The creation of this new field involved re-using a number of the assumptions and abstractions that defined the earlier fields. In this thesis, we show how these abstractions have influenced sensor network protocol designs, note a number of problems with the abstractions, and define new protocols working with improved abstractions.

Abstraction is at the core of everything that we do, precisely because it is at the core of language. What we do, how it is described to others, and what they derive from what we have described are all littered with abstractions. The major reasoning behind the use of abstractions is that there is simply too much information in most situations and an abstraction provides the important details of a situation without burying ourselves in data e.g. thinking of a collection of molecules as a gas, liquid or solid, and considering the collection as a single object, rather than thinking about the individual information about each and every molecule. Furthermore, Reddy [107] and Lakoff [68] showed that our abstractions are built upon common experiences of physical events.

Computer science, especially the software elements of it (which are arguably the vast majority of the field of computer science) is a collection of interrelated abstractions. Many of the physical events associated with abstractions for computing concepts involve one or more people doing a task that we would like the computer to do, e.g. the common abstractions of a "stack" of objects being like a stack of cards; the notion of a "queue" of tasks being like a queue of people; the entire field of "agent-based" computing. For simple examples like queues and stacks, the connection between the abstract concept and the physical example is obvious and clear. Assumptions and extrapolations based on knowledge of characteristics of the physical example have direct analogs in the abstract concept e.g. we can add more people to a queue, remove people from the front of a queue, and the same problems occur with multiple queues (some queues may empty faster than others for example) in both the abstraction and the physical example.

Wireless Sensor Networks (WSNs) are also subject to an abstracted model of thinking, but there is a problem that the abstractions in common use for WSNs have been

copied from the predecessor fields, with little examination or testing as to whether the abstractions are still a valid representation of the actual situation, given the new restrictions of WSNs e.g. power limits. This lack of checking for validity of the abstractions has led to inefficient protocol design when compared to the possibilities with improved abstractions. We therefore set out to show this in a variety of types of WSN protocols, and to build better abstractions for each protocol type.

In Chapter 2, we re-examined MAC protocol design, and came to the conclusion that the standard design specifications were insufficiently abstract, which resulted in MACs that were much larger than required; were unable to reuse parts of the common code that was duplicated; and limited the opportunities for expanding the designs to provide extra capabilities for higher layers, because of the necessity of rebuilding any expansion for every different MAC protocol. We instead described the λMAC framework, which showed how to build MAC protocols that could reuse much larger collections of common code, giving a substantial reduction in the amount of code required to implement a new protocol (e.g. λT-MAC was only 32% of the size of the original T-MAC implementation). This also allows MAC designers to abstract away from problems like time synchronisation, which were previously a major area of difficulty for MAC builders, but with the λMAC framework time synchronisation is done automatically.

In Chapter 3, we looked at routing protocols, and found that the standard primitive for routing protocols (single-hop reliable links) was an energy-inefficient abstraction for WSNs with variable-quality radios. We separated the problems into two groups: the notion of N hops to the sink node was replaced with ETX (Expected TX cost) values, and the single-hop unicast primitive was replaced with ExOR, which better utilises the reality of WSNs as broadcast-based systems. We generalised ExOR to allow for new metrics, as opposed to its original usage of shortest-path hop counts, and delaying metric decisions. From generalised ExOR we created ExOR-ETX using ETX values and ExOR-Bcast for reliable broadcast/flooding of values. We then used these new actions to build Guesswork, an adaptive routing protocol that could work efficiently with a wide variety of link qualities. Guesswork was able to use the delayed metric decisions of generalised ExOR to delay the choice of best next-hop nodes until after packets have reached multiple potential next-hop nodes, which improved reliability with unreliable links.

In Chapter 4, we looked at localisation protocols, and discovered that the primitive information expected by most localisation protocols (unreliable single distance values between pairs of nodes) discarded a lot of information about the actual underlying statistics of the sensors that provided the information. We instead moved to the use of probability maps to better describe mapping between incoming sensor values and the distance between nodes. Using the probability maps, we then built RSL (Refined Statistic-based Localisation), which also used bounding box information based on sanity boundaries for node locations. We also introduced limited broadcast of generated "pseudo" anchors to solve the problems of high processing time and dense anchor formations required by earlier work in statistic-based localisation. We also explored the standard notion of anchor nodes, showed that for most proposed application scenarios that a mobile anchor node could be implemented for similar costs to a single anchor, and that mobile anchors could be used to generate many "virtual" anchors along a path. We showed data for RSL

working with mobile anchors, providing good localisation with the new abstractions.

In Chapter 5, we looked at motion detection. Normally, motion detection would either be done with specialised hardware or with the aid of localisation techniques. Given the probable absence of the former, and that we had redesigned the abstractions for localisation, we explored whether these abstractions could be used to do motion detection. We looked first at working with the probability maps, used them to build force equations for springs in a mass-spring model, and used the mass-spring model to do motion detection for limited amounts of motion. We then showed that the sanity constraints of the location bounding boxes could be exploited to detect larger quantities of motion.

In Chapter 6, we looked at aggregation techniques. We found that the standard problem approached by most aggregation protocols is discovering better ways to distribute various statistical functions (e.g. average, median, count of values exceeding a threshold), and providing ways to manipulate the information for different application scenarios. We discovered that considering standard statistical functions as a usable abstract summary of data from a set of nodes did not match with the requirements of most sensor networks, as most statistical techniques treat rare values as anomalies to be discarded, as opposed to most WSN applications, where the anomalous values are the most important (e.g. "which tree is on fire?"). We instead suggested the notion of aggregation protocols as a "limited information summary", and that "blind aggregation" of data points without considering the values was flawed. We proposed a phase-space representation for sensor data, showed how to merge data sets using this form, and noted when certain sets should not be merged in order to preserve the different data values. This new formation also preserved location data, which was discarded by all previous WSN aggregation protocols. We then described Foxtrot, which uses the phase-space techniques to build a distributed aggregation protocol.

We believe that our study of these five different areas, showing issues across a wide selection of the traditional design problems for sensor networks, is convincing evidence that there were problems with the existing abstractions for WSNs, and that re-examining the abstraction stack for WSNs results in improved abstractions and provides new ways to think about the problems facing future research.

Samenvatting

Sensor netwerken stammen af van verschillende andere onderzoeksgebieden, in het bijzonder gedistribueerde systemen, maar onderscheiden zich door aanvullende randvoorwaarden (oa. minimaal energieverbruik), extra problemen (oa. plaatsbepaling) en nieuwe technieken (oa. source-to-sink routering en data aggregatie). Bij het ontwikkelen van dit nieuwe onderzoeksgebied is veelal gebruik gemaakt van bestaande abstracties en interfaces. In dit proefschrift betogen we dat deze abstracties ongewild de ontwikkeling van sensor netwerken beperkt hebben, analyseren we de onderliggende misvattingen, en presenteren verbeterde abstracties die verwerkt zijn in een reeks nieuwe protocollen voor het efficiënt gebruik van sensor netwerken.

Abstractie ligt aan de basis van alles dat we doen, omdat het een essentieel onderdeel van taal is. Wat we doen, hoe we dat communiceren aan anderen, en wat die daarvan oppikken wordt grotendeels bepaald door het gebruik van abstracties. Dit gebruik is eenvoudig te verklaren door het feit dat een overvloed aan informatie meestal contra productief werkt en dat abstracties voorkomen dat we ons verliezen in de details. Zo helpt het om een verzameling moleculen als één geheel dat wil zeggen een gas, vloeistof of object te beschouwen in plaats van de eigenschappen af te leiden uit de interacties tussen de individuele moleculen. De abstracties die we gebruiken zijn vaak gebaseerd op gemeenschappelijke ervaringen met de fysieke wereld om ons heen [68, 107]

De informatica is in essentie een verzameling samenhangende abstracties. Dit geldt met name voor de software, die in toenemende mate belangrijker wordt dan de onderliggende hardware. Veel van de abstracties die in de informatica gebruikt worden zijn ontleend aan de dagelijkse praktijk en de activiteiten daarin die we graag door een computer zouden laten uitvoeren. Bijvoorbeeld, het concept “queue” is naar analogie van een rij mensen wachtend voor de kassa, het begrip “stack” komt overeen met een stapel kaarten, en het hele onderzoeksveld van agent systemen heeft een duidelijke parallel met de functies die een reisagent voor ons verleent. Kennis van de fysieke eigenschappen en natuurkundige wetten die gelden in de wereld om ons heen laten zich vaak letterlijk vertalen naar de abstracte concepten die we gebruiken in de informatica. Rijen worden langer als er meer mensen moeten wachten, degene die vooraan staat is het eerst aan de beurt, en als we moeten kiezen uit meerdere rijen is er altijd het probleem dat de ene sneller geholpen wordt dan de ander.

Ook in Wireless (draadloze) Sensor Netwerken (WSN) wordt er ruimhartig gebruik gemaakt van abstracties om de complexiteit van zulke grootschalige systemen het hoofd te bieden. Het probleem echter is dat de abstracties veelal rechtstreeks overgenomen zijn uit reeds ontwikkelde systemen en methodieken zonder af te vragen en/of te verifiëren of ze nog steeds van toepassing zijn gegeven de nieuwe randvoorwaarden in het WSN domein. Dit klakkeloos gebruik van oude abstracties kan leiden tot inefficiënties die vermeden zouden kunnen worden door de juiste abstracties te gebruiken. In dit proefschrift hebben we daarom een breed scala aan WSN protocollen, en onderliggende abstracties, onderzocht en waar mogelijk aangepaste abstracties geïntroduceerd om te komen tot betere protocollen.

In hoofdstuk 2 hebben we kritisch gekeken naar hoe de huidige MAC protocollen voor WSN gestructureerd zijn, en geconcludeerd dat er te weinig gebruik gemaakt is van gemeenschappelijke abstracties waardoor functionaliteit zoals time synchronisation in ieder MAC protocol opnieuw geïmplementeerd is. Het gebrek aan abstractie bemoeilijkt ook het uitbreiden van bestaande protocollen met nieuwe functionaliteit ten behoeve van hogere lagen in de protocol stack omdat dat impliceert dat alle MAC's afzonderlijk aangepast moeten worden. We hebben daarom het λMAC framework gedefiniëerd dat laat zien hoe MAC protocollen geïmplementeerd kunnen worden middels een gelaagde, interne structuur die het ontwikkelen van nieuwe MAC protocollen aanzienlijk versnelt omdat grote delen van de framework implementatie hergebruikt kunnen worden. Zo is de λT-MAC implementatie slechts 32% van de grootte van de originele T-MAC code. Verder kunnen MAC ontwikkelaars op een hoger abstractie niveau werken en hoeven zich niet meer bezig te houden met zaken als time synchronisation, iets dat berucht is vanwege de moeilijk traceerbare race condities die hier vaak de kop opsteken.

In hoofdstuk 3 hebben we routeringsprotocollen bestudeerd en geconstateerd dat de veelgebruikte basis abstractie van een betrouwbare unicast link alleen tegen hoge kosten (energieverbruik) aangeboden kan worden door de onderliggende MAC laag vanwege de grote variatie in kanaal kwaliteit die WSN radio's karakteriseren. Onze oplossing is tweeledig. Ten eerste stappen we af van de notie van *N* hops, en maken gebruik van de ETX (Expected TX) metriek die rekening houdt met het verwachte aantal retransmissions. Ten tweede, vervangen we de single-hop unicast communicatie door ExOR stijl communicatie die wel gebruik maakt van de broadcast eigenschappen die het medium radio biedt. Om deze broadcast primitief optimaal te benutten hebben we hem aangepast zodat hij ook toepasbaar is op andere metrieken dan de originele hop-count, en andere beslis-momenten toelaat. In het bijzonder hebben we twee nieuwe communicatie mechanismen geïntroduceerd: ExOR-ETX en ExOR-Bcast. De laatste implementeert flooding, een operatie die in allerlei routeringsprotocollen gebruikt wordt naast het versturen van applicatie data. Door ExOR-ETX en ExOR-Bcast te combineren in een nieuw protocol, genaamd Guesswork, zijn we erin geslaagd een routeringsprotocol te ontwikkelen dat zich moeiteloos aanpast aan fluctuerende kanaal kwaliteiten. Een van de sleutels tot succes is dat Guesswork ExOR gebruikt om de beslissing naar welke buur node de boodschap te sturen uit te stellen tot het moment dat alle kandidaten de gelegenheid gehad hebben een bevestiging (ACK) te sturen. Dit maakt het mogelijk om onder gunstige omstandigheden direct met verre buren te communiceren in plaats van altijd conservatief te

kiezen voor een naaste buur die altijd met hoge waarschijnlijkheid te bereiken is. Deze opportunistische strategie laat toe om het aantal hops te verminderen en energie te besparen als de omstandigheden zich daartoe lenen.

In hoofdstuk 4 hebben we plaatsbepalingsalgoritme onder de loep genomen. Veel van deze algoritmen zijn gebaseerd op het gebruik van éénmalige afstandsschattingen tussen paren van nodes. Echter de onbetrouwbaarheid van deze metingen en de verdeling daarvan wordt niet meegenomen waardoor de behaalde nauwkeurigheid van de berekende posities veel te wensen over laat. Wij hebben daarom geëxperimenterd met een nieuwe abstractie tussen afstandssensor, in veel gevallen de radio, en plaatsbepalingsalgoritme: in plaats van één getal gebruiken we een waarschijnlijkheidsverdeling. Door expliciet met deze verdelingen te "rekenen", in combinatie met het gebruik van een mobiel referentie punt (anchor node) dat vele meetpunten beschikbaar maakt, zijn we erin geslaagd een plaatsbepalingsalgoritme te ontwikkelen dat nauwkeuriger resultaten bereikt. Dit RSL (Refined Statistics-based Localastion) algoritme is verder aangepast om ook in scenario's met weinig afstandmetingen te kunnen opereren middels het introduceren van 'pseudo anchors' en rekening houden met aanvullende eisen op basis van het maximale radio bereik (bounding-box constraints). Het grootste voordeel is echter behaald met de abstractie van afstandswaarschijnlijkheidsverdeling.

In hoofdstuk 5 hebben we het aanverwante probleem van beweginsdetectie beschouwd. Men kan gebruik maken van speciale hardware in de vorm van versnellingsopnemers, maar uit kosten oogpunt is het veel aantrekkelijker om deze informatie indirect af te leiden uit opeenvolgende plaatsbepalingen. Ook in dit geval blijkt het voordelen te hebben om met afstandswaarschijnlijkheidsverdelingen te werken, en we hebben laten zien dat deze effectief gebruikt kunnen worden in mass-spring systemen om kleine bewegingen te detecteren. De bounding-box constraints komen van pas om grotere bewegingen te detecteren (een inconsistentie duidt op een aanzienlijke verplaatsing).

In hoofdstuk 6 tenslotte, hebben we data-aggregatie algoritmen bestudeerd. De overgrote meerderheid richt zich op het efficiënt implementeren van standaard statistische bewerkingen (gemiddelde, mediaan, histogram, enz.) in diverse WSN toepassingen. Echter, de impliciete veronderstelling dat zulke statistische informatie relevant is voor deze toepassingen is in onze optiek incorrect: juist de uitschieters in gemeten sensor waarden zijn interessant ("Welke boom staat in brand?"), terwijl de statistische bewerkingen deze met opzet negeren ("De gemiddelde temperatuur is 20,7 graad C."). Deze observatie heeft ons gemotiveerd een nieuwe abstractie voor data aggregatie te introduceren, de phase-space representatie, die wel rekening houdt met de relevantie van uitschieters en tevens de positionering van de diverse sensor waarden expliciet in ogenschouw neemt. We hebben vervolgens een nieuw data aggregatie protocol (Foxtrot) geïmplementeerd dat met behulp van een verzameling applicatie-specifieke regels in het netwerk beslist welke phase-space's geaggregeerd kunnen worden ("temperatuur met de nauwkeurigheid van een halve graad") en welke niet ("spatiële resolutie van minimaal 10m"). De resultaten van Foxtrot zijn in vergelijking tot bestaande aggregatie protocollen bemoedigend: de waardevolle uitschieters worden afzonderlijk gerapporteerd, terwijl vergelijkbare waardes in het netwerk geaggregeerd worden en er dus weinig extra data-verkeer benodigd is.

Wij concluderen op basis van de vijf aspecten van sensor netwerken die we in detail bestudeerd hebben, dat de hergebruikte abstracties inderdaad niet de juiste zijn en voor problemen zorgen zowel wat betreft de efficiëntie van de protocollen die daarop gebaseerd zijn, als de aangeboden functionaliteit die niet overeenkomt met de beoogde toepassingen. Een herziening van deze abstracties is dus op zijn plaats en het onderzoek beschreven in dit proefschrift laat zien dat hiermee voordelen behaald kunnen worden en opent de weg naar verder onderzoek.

Curriculum Vitae

Tom Parker was born in London, England on January 8, 1981. He received his B.Sc. (Hons) in Computer Science from the University of Bristol in 2003. Later that year, he in parallel started both his M.Sc. and Ph.D. in the Parallel and Distributed Systems group at Delft University of Technology. In 2005, he completed his M.Sc., with his thesis on Localisation in Mobile Anchor Networks. His research interests include routing protocols, data aggregation models, sensor node software architectures, and finding ways to change how we think.

He is currently employed as a researcher in the Parallel and Distributed Systems group at Delft University of Technology.

Selected Publications

1. T. Parker and K. Langendoen. Refined statistic-based localisation for ad-hoc sensor networks. In *IEEE Workshop on Wireless Ad Hoc and Sensor Networks (associated with Globecom 2004)*, Dallas, TX, Nov. 2004.

2. T. Parker and K. Langendoen. Guesswork: Robust routing in an uncertain world. In *2nd IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2005)*, Washington, DC, Nov. 2005.

3. T. Parker and K. Langendoen. Adumbrate: Motion detection with unreliable range data. In *4th Int. Workshop on Networked Sensing Systems (INSS 2007)*, pages 221–228, Braunschweig, Germany, June 2007.

4. T. Parker and K. Langendoen. Foxtrot: phase space data representation for correlation-aware aggregation. In *Fourth IEEE Conf. on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON)*, San Diego, CA, June 2007.

5. T. Parker, M. Bezemer, and K. Langendoen. The λMAC framework: redefining MAC protocols. PDS Technical Report PDS-2007-004, Delft University of Technology, Sept. 2007.