Energy Efficient Duty Allocation Protocols For Wireless Sensor Networks

Jonathan Tate and Iain Bate
Department of Computer Science
University of York
York, United Kingdom, YO10 5DD
Email: { jt | iain.bate }@cs.york.ac.uk

Abstract

Wireless sensor networks require shared medium access management to prevent collisions, message corruption and other unhelpful effects. Cellular sensornets require minimal energy consumption to maximise network lifetime, and management of interaction with base stations and other cells. We present a protocol which dynamically generates a near-optimal duty schedule within a cell such that communication duty is shared evenly between participating nodes with exactly one node on-duty at any given time.

1. Introduction

Sensornets compose many small, low-cost computing nodes into distributed systems deployed into physical environments of interest. Nodes have restricted energy, computation and storage resources and therefore limited utility in isolation; co-operation and coordination is necessary to address realistic problems.

Consider a large sensornet consisting of many nodes, divided into cells containing smaller numbers of nodes in close geographic proximity [1]. Within a cell each node has a similar view of the physical environment, and similar connectivity to nearby base stations or surrounding cells [2]. It follows that all nodes within a cell are approximately equivalent with respect to extracellular entities and environmental context.

Suppose that an external entity broadcasts a message received by all members of a cell. Unless the message is intended for a specific member of that cell, it is unclear which cell member or set of cell members should respond. Data packets to be forwarded to remote destinations need only be rebroadcast once; if all cell members rebroadcast this wastes energy, increases contention for the wireless medium, and risks collisions [3]. If a tasking message requests that a sample value be read from the physical environment then all cell members will produce equivalent readings [4]. Consequently, energy and network capacity may be wasted in delivering multiple redundant messages.

Any of a number of similarly positioned nodes are equally valid candidates to handle specific tasks. Some mechanism is required to avoid wasteful repetition, and mitigate ambiguous or unpredictable multiple responses to stimuli, by enforcing mutual exclusion [5].

By deterministically assigning responsibility for response, we implicitly identify the nodes which will not be required to respond. These nodes can switch unused energy-hungry subsystems into low power modes. The consequent energy saving extends the useful lifetime of sensornets composed of nodes with finite energy resources. Sensornets can run indefinitely if duty cycle allocation allows nodes to scavenge energy from the environment at the rate of consumption [6].

The *Cyclic Duty Allocation Protocol* (CDAP) is an application- and platform-agnostic lightweight protocol to cycle duty between the nodes of a network cell. System *epochs* are divided into portions of equal length and allocated fairly among nodes, such that each node is assigned responsibility for one portion during each epoch. Exactly one node is deterministically assigned this responsibility at any arbitrary time, removing ambiguity as to which node must respond to stimuli. Applying well-understood *synchronisation* phenomena observed in nature [7], inter-node coordination is achieved by cells acting as closed systems of pulse-coupled oscillators. As cells approach stable equilibrium states, nodes can identify periods in which energy-saving states can safely be entered.

The remainder of this paper is structured as follows. Section 2 considers related work. Section 3 defines the contributions of this paper. Section 4 specifies the requirements and design of the protocol. Section 5 considers protocol extensions that improve energy efficiency. Section 6 compares experimental results to theoretical expected performance. Finally, section 7 summarises conclusions to be drawn from this work.

2. Related work

In a typical sensornet it is rare for all nodes to perform useful work at all times. Energy efficiency can be improved by carefully managing node state, placing some subset of the network in low-energy inactive states when not required to actively participate [8]. However, finding the optimal sleep schedule requires global knowledge of all node tasks and schedules to be maintained. Sensornets generally have insufficient resources to support the communication, computation and storage overheads of these optimal algorithms, with energy cost exceeding the resultant savings [9].

Low-level approaches minimise energy costs by identifying periods during which node subsystems are not fully utilised [10]. If components consume less energy when running at less than 100% capacity it is often beneficial to off-load activity from busy periods to less-busy periods, or to work speculatively in idle periods to minimise periods running at 100% capacity.

The *Random Asynchronous Wakeup* protocol [2] implements a randomised and distributed algorithm under which nodes make local decisions on whether to sleep or remain awake. Within each time frame each node is awake for a randomly chosen fixed interval. When forwarding packets an integrated routing protocol selects from a set of equivalent next-hop locations with probabilistic guarantees that at least one of these will be awake. However, as there is no coordination between nodes there is no guarantee that any forwarding candidates are awake, and if more than one is awake this redundancy wastes energy.

Similar functionality is provided by the *Asynchronous Random Sleeping* scheme [11] which is principally useful where no inter-node coordination is possible. However, such scenarios might be considered unusual as sensornets generally execute distributed and cooperative sensing and processing applications.

The *Probing Environment and Adaptive Sleeping* protocol [5] implements an adaptive sleep policy in which nodes sleep for an exponentially distributed duration then wake and transmit a probe message. If any nearby nodes happen to be awake they transmit a reply message. If any such reply message is received the node is not required at this time and sleeps again; otherwise, it remains awake until it fails or runs out of energy. A significant weakness is that when a node fails there is zero local network coverage until some other nearby node wakes with indeterminate delay. If the failed node has accumulated significant data this cannot be replaced by that of other nearby nodes.

The *Lightweight Deployment-Aware Scheduling* algorithm [12] aims to improve network energy efficiency by switching off redundant nodes without access to accurate location or directional information. Observing that nodes require up to 11 active neighbours to provide a 90 percent chance of complete redundancy, LDAS allows network designers to trade-off sensing redundancy against energy consumption. This protocol is most appropriate and efficient in networks where most nodes are required for physical sensing rather than for distributed data processing.

An alternative view [13] is that application-aware traffic scheduling, rather than network coverage, holds the key to maximising energy efficiency and hence network lifetime. The *Multi-Sensornets* approach applies a genetic algorithm to balance nodes' energy consumption in a distributed data fusion application. Peer nodes coordinate dataflow schedules such that the time during which they are required to be awake is minimised, allowing nodes to safely sleep at other times without disrupting network coverage.

Synchronisation and *desynchronisation* are biologically-inspired primitives in which a closed finite system of periodic oscillators converge to a steady state. Under *synchronisation* all oscillators fire simultaneously in the steady state [7], whereas under *desynchronisation* the oscillator firing times are evenly distributed in time in the steady state [14]. In a *synchronised* system a set of two or more periodic isochronal events are coordinated to begin and end at the same time. By contrast, in a *desynchronised* system these events are organised to maximise the inter-event period, which is equal for all pairs of events and their immediate successors. The desired system-wide coordination is an emergent property of independent agents implementing simple rules [7].

3. Contributions

We define the following research objectives as the primary contributions of this paper.

- Obj 1: Define a lightweight duty allocation protocol for unicellular sensornets such that exactly one node is active at any given time.
- Obj 2: Evaluate the duty allocation protocol experimentally and theoretically to assess its efficacy and energy efficiency.

4. Cyclic Duty Allocation Protocol

The *desynchronisation* primitive generates a periodic sequence of synchronisation transmissions spaced evenly in time. We use this as the basis of our protocol.

4.1. Desynchronisation

Assume a network cell consists of a set Σ of nodes $S_1 \cdots S_n$ where $n \geq 2$; if $n = 1$, there is obviously no need for inter-node coordination. Each node S_i acts

independently but shares an identical set of behavioural rules. The running time of the system is divided into a set of system *epochs* of equal period e such that $\forall j : E_j = e$. The sequence of system epochs E_j is defined by the natural ordering of $j \in \mathbb{N}$.

Within each system epoch E_j it is required that each node $S_i \in \Sigma$ shall execute a single instance of a periodic synchronisation event V_i exactly once. These events are used only by the protocols described in this paper, and are not related to any events used by the sensornet application. All events V_i are periodic with identical period $p_i = e$. The occurrence of a specific event at a specific node i within a specific system epoch j is labelled V_{ij}. It is required that all events V_{ij} are executed within epoch E_j.

Each node $S_i \in \Sigma$ has a local clock used to measure the *local phase* ϕ_i which increases from 0 to ϕ_{max} in time $p_i = e$. When $\phi_i = \phi_{max}$ at node S_i in epoch E_j, node S_i transmits a *synchronisation message* and resets its local phase as $\phi_i = 0$, corresponding to event V_{ij}. No global clock is required. Peer nodes $T \in (\Sigma \setminus S_i)$ receive the V_{ij} synchronisation message at their local phase ψ_{ij} but do not know the identity of S_i.

Within every epoch E_j all nodes S_i record the local phase of peer node synchronisation messages, using this information to modulate their local phase to coordinate behaviour within the cell [7]. *Desynchronisation* protocols maximise the time between synchronisation events for all nodes in a given epoch, converging on an *equilibrium state* in which synchronisation events occur spaced evenly in time [14].

Each node S_i records the local phase of the peer synchronisation events $V_{i\beta}$ and $V_{i\gamma}$, occurring immediately before and immediately after the local synchronisation event V_i respectively, and discarding all others. The corresponding peer nodes $S_{i\beta}$ and $S_{i\gamma}$ are labelled the *phase neighbours* of node S_i. The phase difference between V_i and $V_{i\beta}$ is calculated as $\phi_{i\beta}$, and the phase difference between V_i and $V_{i\gamma}$ is calculated as $\phi_{i\gamma}$. Note that $\phi_{i\beta}$ is always negative and $\phi_{i\gamma}$ always positive owing to natural ordering of events.

The *phase error* $\theta_i = \phi_{i\beta} + \phi_{i\gamma}$ is the phase difference between the local synchronisation event V_i and the target midpoint of phase neighbour synchronisation events $V_{i\beta}$ and $V_{i\gamma}$. Each node S_i alters its local phase by $\Delta\phi_i = -f_\alpha\theta_i$ upon observing $V_{i\gamma}$, where $f_\alpha \in (0, 1]$ is the *feedback proportion* governing the balance between responsiveness and stability. Given an otherwise unchanging network, $\forall i : ||\Delta\phi_{ij}|| \to 0$ as $j \to \infty$ in successive epochs [7]. Observe that $\forall i : \theta_i = 0$ in the *desynchronised equilibrium state*. If phase error $\theta_i = 0$ then phase change $\Delta\phi_i = 0$ also; no action is required when the system has converged.

4.2. Synchronisation transmissions

Networks in which this primitive is applied can be modelled as a fully connected graph $\mathcal{G} = (\Sigma, \mathcal{E})$, where Σ represents the set of network nodes and $\mathcal{E}$ represents the set of possible pairwise communication exchanges.

The events V_i executed by nodes $S_i \in \Sigma$ as described above are short *pulses* which are broadcast by a sender node S_α and received by all other nodes $S_i \in (\Sigma \setminus S_\alpha)$. The edges $\mathcal{E}$ of the graph $\mathcal{G}$ can be thought of as representing communication channels which are often unused, but through which a single bit of information will periodically be transmitted when a *pulse* transmission occurs.

A typical implementation would be the smallest valid packet achievable within a given network stack; the packet *data* is greater than one bit but conveys one bit of *information*. The minimal time required for this stub packet to traverse the network stack of the sender and the receivers, κ, represents the limit of convergence of the *desynchronisation* primitive.

Assuming cells contain n nodes the minimal overhead per epoch is $n\kappa$. For epochs of length e the proportion p of each epoch available for application data transmission is $p = 1 - (n\kappa/e)$. Note that $p \to 1$ as $e \to \infty$; longer epochs imply smaller overhead but greater stabilisation time as per section 4.1.

Recipients will use the time at which the *pulse* is received, rather than information encoded into the signal itself, as the source data for the coordination algorithm. Other protocols could encode additional information within these transmissions at the expense of higher overhead, but we do not require this.

4.3. Protocol states

A simple Finite State Machine runs at each node. The states define the communication responsibilities of a given node at a given time with regard to peer nodes within the cell and external entities beyond the cell. As the local phase ϕ_i of node S_i increases from 0 to ϕ_{max} the protocol state may be changed by detected synchronisation events, or by state timeouts.

ONDUTY - Node is responsible for communications duties of the cell, and is responsible for handling any incoming packets. Node can hear both application messages and synchronisation messages. Radio modules are switched on and ready for bidirectional exchange with neighbouring entities, and transmit their own synchronisation message in the middle of this period.

SCAN - Node is listening for synchronisation messages but has not yet collected sufficient data to predict times of phase neighbour peer node synchronisation transmissions. Node can hear syn-

chronisation messages but do not expect application messages. Node radio module is switched to the lowest-power mode that can detect the synchronisation messages, except when transmitting its own synchronisation message.

SYNC - Node is waiting for synchronisation messages around the predicted times of phase neighbour peer node synchronisation transmissions. Node can hear synchronisation messages but do not expect application messages. Node radio module is switched to the lowest-power mode that can detect the synchronisation messages.

OFFDUTY - Node has no communication responsibilities and is free to switch radio modules off or into other low-power modes. Nodes can hear neither application nor synchronisation messages.

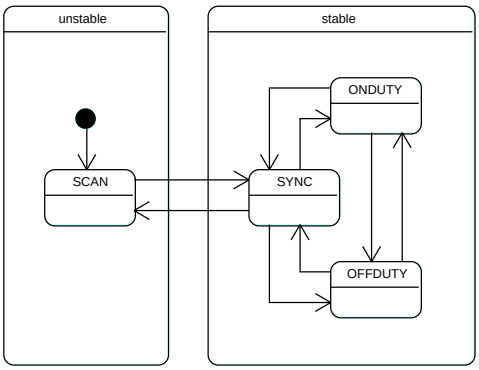

Figure 1. CDAP States

Figure 1 illustrates the CDAP states and state transitions in UML statechart format. All nodes start in *SCAN* during which each node S_i collects timing data about phase neighbour synchronisation events $V_{i\beta}$ and $V_{i\gamma}$. This continues for at least time $t = 2e$ to ensure that all nodes observe at least one occurrence of each phase neighbour event, hence allowing each node to produce at least a first approximation estimate of the next instance of these events. Upon reaching this point nodes transition from the *unstable* composite state to the *stable* composite state, entering the latter in the *SYNC* simple state.

The CDAP protocol builds a distributed schedule which defines the periodic duty cycles for each node in each system epoch. We will consider the mechanism by which this schedule is constructed in section 4.4 but these details are irrelevant at this point. The duty period in state *ONDUTY* is obtained for each node, with that node's own synchronisation transmission occurring at the midpoint of the *ONDUTY* period and the phase neighbours' synchronisation transmissions occurring at times outside the *ONDUTY* period. In the most basic form of CDAP it follows that by default nodes are in the *SYNC* state listening for synchronisation messages, transitioning temporarily to *ONDUTY* when the duty period begins and transitions back to *SYNC* when the duty period ends.

This provides the desired mutual exclusivity property in which there is exactly one node in the *ONDUTY* state at all times except for a small handover period between nodes. The *ONDUTY* state takes precedence over all others if timing errors lead to conflict at any node. However, this basic schedule is not energy efficient. Listening to the wireless medium for synchronisation messages often consumes energy as quickly as listening for application messages, and always consumes more energy than a low-power standby state. We would prefer nodes to spend time in the *OFFDUTY* state to conserve energy when possible.

We address this issue by observing that phase neighbours' synchronisation transmissions are generally short in comparison to the system epoch, and occur at reasonably predictable times as the system converges on the equilibrium state if signal noise and timing error is moderate. For example, transmission times are subject to jitter with magnitude that is usually (but not always) small compared to epoch length e. Imperfect node clocks will drift out of synchronisation such that relative phase offset of phase neighbour synchronisation events will inevitably change over time if the protocol is not continually adaptive.

It is therefore sufficient to limit listening in the *SYNC* state to relatively small *synchronisation windows* during which there is a reasonable expectation, though no guarantee, that peer nodes will transmit their synchronisation messages. The details are irrelevant at this point but are defined in section 5.2. In stable systems predictions will usually be reasonably accurate; prediction failures can be handled as and when they arise with lower total cost than the basic policy.

Participating nodes can therefore employ *OFFDUTY* as the default state rather than *SYNC*. Nodes transition from *OFFDUTY* to *SYNC* shortly before the predicted synchronisation transmission from the preceding phase neighbour, and then back to *OFFDUTY* shortly afterwards. Some time passes in *OFFDUTY* until the assigned duty period begins, at which point the node transitions to *ONDUTY* until the duty period ends and the node transitions back to *OFFDUTY*. A further transition to and from *SYNC* occurs around the predicted synchronisation transmission from the succeeding phase neighbour. The node is then able to remain in *OFFDUTY* until the next predicted time of the preceding phase neighbour synchronisation event. This cyclical pattern of transitions repeats with the

same periodicity as the system epoch, e.

It is mentioned above that synchronisation events are not guaranteed to occur at the predicted times. This may be due to failure of the node which was due to transmit its synchronisation message, signal noise at the receiver, timing error at the transmitter or receiver, or an overly optimistic truncation of the *SYNC* state time. The protocol determines if sufficient timing data exists to make predictions in subsequent system epochs. If yes, the protocol proceeds as before, and is hence able to reject small or transient errors. If no, the node transitions from the *stable* composite state to the *unstable* composite state, and the *SCAN* simple state in particular. The protocol then restarts listening for peer nodes' synchronisation messages and transmitting its own synchronisation messages at the scheduled times.

This allows a given node to temporarily drop out of active service, without adversely affecting the cell's other nodes or the distributed schedule, rejoining soon after. If a node fails completely its disappearance will be noted in the same way, but as it will no longer transmit synchronisation messages the distributed schedule will eventually reconverge on a new equilibrium state.

4.4. Allocating duty periods

The *desynchronisation* primitive described in section 4.1 obtains an equilibrium state in which a sequence of synchronisation events is evenly distributed throughout time. We now use these synchronisation events to allocate *ONDUTY* state periods. We use a method similar to that employed by the *DESYNC-TDMA* protocol [14] to mediate access to a shared wireless medium.

Recall that $\forall i : ||\Delta\phi_{ij}|| \rightarrow 0$ as $j \rightarrow \infty$ in successive epochs. As the system converges on the desynchronised equilibrium state, at each node S_i the phase differences $\phi_{i\beta}$ and $\phi_{i\gamma}$ between the local synchronisation event v_i and the phase neighbour synchronisation events $V_{i\beta}$ and $V_{i\gamma}$ will converge on $-\phi_{max}e/2n$ and $+\phi_{max}e/2n$ respectively.

It follows that, as the system converges toward the equilibrium state, each node can predict the time of its phase neighbours' synchronisation events with increasing accuracy. This is important as we must predict the timing of successor phase neighbour event $V_{i\gamma}$ from historical values in order to allocate duty periods that extend beyond the local synchronisation event V_i. Otherwise, we must end duty periods at the occurrence of V_i, preventing allocation of more than 50% of each system epoch to active duty among nodes.

We define the duty period of node S_i in terms of phase offsets of phase neighbours' synchronisation events $V_{i\beta}$ and $V_{i\gamma}$, measured from the local synchronisation event V_i at local phase $\phi_i = \phi_{max}$. The duty period starts halfway between the occurrence of $V_{i\beta}$ and V_i, and ends halfway between the occurrence of V_i and $V_{i\gamma}$. To reduce the risk of two nodes' duty periods overlapping as a consequence of clock error we can scale down the duty period length l to ηl. Higher values of the scaling constant $\eta \in (0, 1]$ give longer duty periods and a greater proportion of each epoch allocated to active duty, but less unallocated inter-periodic buffer time.

Consider node S_i. Recall from section 4.1 that $\phi_{i\beta}$ is the phase offset of $V_{i\beta}$ and $\phi_{i\gamma}$ is the phase offset of $V_{i\gamma}$. The p most recent measured values of each phase offset are retained, a *null* value being stored if an expected synchronisation event is not observed. Predicted values of $\phi_{i\beta}$ and $\phi_{i\gamma}$ are taken as moving averages over historical values to reject timing noise.

There are *sufficient* measurements to predict timings of phase neighbours' synchronisation events if at least proportion q of the p stored values are non-null and no more than r consecutive null values have been stored, ensuring predictions are based on acceptably complete and timely observations. If *insufficient* historical values have been collected, the node must re-enter the *SCAN* state to capture more prediction data as per section 4.3.

The duty period starts at phase $\phi_i = -\lambda||\phi_{i\beta}||$ and stops at phase $\phi_i = +\lambda||\phi_{i\gamma}||$. As phase $\phi \in [0, 1)$ we apply modular arithmetic to convert the start phase value to the equivalent $\phi_i = \lambda||\phi_{max} + \phi_{i\beta}||$. We set the scaling constant $\lambda = (1 + \eta)/2$ to split the unallocated buffer time defined by η between the beginning and end of the duty period.

As the system converges on the equilibrium state defined in section 4.1 the start and stop phase offsets of duty periods will converge on $-\lambda\phi_{max}e/2n$ and $+\lambda\phi_{max}e/2n$ respectively. However. we cannot simply use these convergence limits from the outset as the protocol must align the local phase of each node with that of its phase neighbours, and hence indirectly with all nodes in the cell. The cell population n is not necessarily known by any node owing to the vagaries of initial deployment, node failures, or cell population changes. Furthermore, before convergence the relative phase of synchronisation events is in flux.

In section 4.3 we state exactly one node is *ONDUTY* at any given time, and is implicitly responsible for sending application packets if an immediate response to observed stimuli is required. Nodes in other states which create application packets must wait $\leq e/n$ time units to regain the *ONDUTY* state before transmitting.

4.5. Measuring effectiveness

CDAP is a state management protocol but has implications for packet routing. Sensornets are composed

of unreliable nodes deployed into hazardous environments. It is therefore inappropriate to route application packets by unique node identifier; some sensornets do not allocate globally unique identifiers and any individual node along delivery routes may fail.

A data-centric geographic routing policy is thus appropriate, in which packet routing decisions are based on node physical location rather than logical network topology. As packets are routed between physical locations, and there is no guarantee that any live node is located at the exact specified destination, it follows that any node sufficiently near the specified destination is equally acceptable. CDAP determines which redundant candidate actually takes responsibility, independently of the packet content or application type.

The rôle of CDAP is to construct and dynamically maintain a duty schedule such that exactly one node is in the *ONDUTY* state at any given time. If zero nodes are in the *ONDUTY* state then communication between the cell and external entities will fail. If two or more nodes are in the *ONDUTY* state then it is undefined which is responsible for external communications.

CDAP can be extended to support delivery between uniquely identified nodes, or to support more complex coverage models in which multiple redundant nodes must be active at all times within cells, but we do not include details here owing to lack of space.

We define the following metrics: P_0, P_1 and P_2. The sensornet executes the protocol as runtime t increases in the interval $[0, \infty]$. Each metric $P_0 - P_2$ measures the proportion of time during a measurement period $[t_{start}, t_{stop}]$ which a given number of nodes are in the *ONDUTY* state.

- P_0: Proportion of time in which zero nodes are in the *ONDUTY* state. Unitless. Defined in the range $[0, 1]$. The ideal value of $P_0 = 0$.
- P_1: Proportion of time in which one node is in the *ONDUTY* state. Unitless. Defined in the range $[0, 1]$. The ideal value of $P_1 = 1$.
- P_2: Proportion of time in which two or more nodes are in the *ONDUTY* state. We do not record the exact number of such nodes, only that there are ≥ 2. Unitless. Defined in the range $[0, 1]$. The ideal value of $P_2 = 0$.

CDAP is a dynamic protocol and hence requires some time to stabilise, attaining the equilibrium state at time t_{eq} as described in section 4.1. The system will remain within this steady state until the network changes, for example where a node joins or leaves the network cell. If $t_{start} \geq t_{eq}$ then all measurements are taken in the equilibrium state, and the values of $P_0 - P_2$ will approximate the theoretical optimal values given below. The longer the measurement period $p = t_{stop} - t_{start}$, the better the approximation as the influence of measurement granularity diminishes.

If, however, $t_{start} < t_{eq}$, the measured values of $P_0 - P_2$ will be influenced by the stabilisation period of sub-optimal behaviour prior to the system reaching equilibrium state at t_{eq}. Although this accurately reflects network performance during the measurement period, it does not necessarily reflect the long-term stable performance as the influence of this pre-*equilibrium* period becomes insignificant as $t \to \infty$. Both measurement scenarios are correct and useful but must be interpreted appropriately; the former describes the long-term stable behaviour, and the latter describes the short-term behaviour during initialisation.

5. Energy efficiency

CDAP switches off radio modules when nodes are not on duty. Other components such as CPU, memory or sensors may optionally be powered down if this is compatible with application requirements. *Synchronisation windows* suppress this during peer synchronisation transmissions to ensure correct CDAP behaviour.

5.1. Radio module states

We define an abstract model of sensornet radio models in terms of a finite set of permitted states. We assess the energy efficiency of a sensornet based on a specific hardware platform by binding a specific power value to each defined radio module state, and measuring the time spent in each state over the runtime of a sensornet. From these measurements we can trivially calculate the energy consumed in each radio module state, and hence the average energy consumption rate for a participating sensornet node. We assume antenna gain and transmit power is fixed for all transceivers.

STANDBY - Low power mode in which nodes can neither transmit nor receive.

LISTENLOW - Low power mode in which nodes can detect nearby transmissions but not receive data.

LISTEN - Node is listening to wireless medium but not currently receiving data.

RECEIVE - Node is listening to wireless medium and currently receiving data.

TRANSMIT - Node is transmitting into the wireless medium.

Transitions between permitted states is controlled by the CDAP protocol. The lowest power state which supports required functionality is selected.

Nodes in *OFFDUTY* keep radio modules in *STANDBY*. Nodes in *SCAN* keep radio modules in *LISTENLOW*, switching temporarily to *TRANSMIT* to transmit synchronisation messages. Nodes in *SYNC* keep radio modules in *LISTENLOW*. Nodes in *ONDUTY* keep radio modules in *LISTEN*, switching

temporarily to *RECEIVE* when receiving application messages or to *TRANSMIT* when transmitting synchronisation or application messages.

If a given hardware platform does not explicitly support an abstract model state we substitute the lowest cost supported state that provides the same functionality. For example, some hardware platforms support *LISTENLOW* in which nodes cannot exchange data but can detect transmissions, which is sufficient for synchronisation. If *LISTENLOW* is not available then *LISTEN* is substituted, switching temporarily to *RECEIVE* when receiving synchronisation messages.

In *LISTENLOW* nodes cannot inspect packet contents to differentiate between application packets and unexpected synchronisation packets. The decision can be made using timing data. In section 4.2 we specify that synchronisation packets are as short as possible; longer transmission times imply application packets.

5.2. Window management

We cannot predict the actual times of synchronisation events with certainty. We can, however, give probabilistic guarantees that they will occur within defined finite periods. We exploit this fact by limiting costly wireless activity to these periods. In this section we describe mechanisms by which the length of the synchronisation windows is gradually reduced as the system converges on the desynchronised equilibrium state toward a final state in which the synchronisation window length reaches a specified minimum.

Within the duration of the system epoch each node has two synchronisation windows; one pertaining to the predecessor phase neighbour synchronisation event, and the other pertaining to the successor. Note that although both window lengths are likely to reduce simultaneously there is no guarantee that this will happen. For example, a given node might transmit its synchronisation event with abnormally high jitter, or might be poorly positioned in the wireless medium landscape and hence frequently fail to be heard by its phase neighbours. We therefore track the predecessor and successor synchronisation windows separately.

It is permitted for these synchronisation windows to overlap; this overlap has no effect as both simply place a node into the *SYNC* state to listen for synchronisation transmissions of which all occurrences are equivalent.

The actual time of a synchronisation event may be earlier or later than the predicted time with equal likelihood as a result of the desynchronisation primitive described in section 4.1. We also assume that transmission time jitter and timing errors from imperfect clocks is equally likely to be positive as negative. We therefore define the synchronisation window of length μ phase units as centred on the predicted synchronisation event time, extending symmetrically by $\mu/2$ phase units in either direction.

If the synchronisation pulse requires time κ (see section 4.2) then we restrict μ to the interval $[2\kappa, \phi_{max}]$ to prevent the window length becoming smaller than the transmission length κ with a reasonable safety margin, and to prevent the window length becoming longer than the system epoch length e. When μ reaches the 2κ threshold, and remains there during subsequent epochs, we measure the steady state energy profile.

We provide estimates of the proportion of time nodes spend in CDAP states defined in section 4.3. For a system converged on the equilibrium state we know that the proportion of time spent in *SCAN* is 0, and the proportion of time spent in *ONDUTY*$= 1/n$ so we need not consider these further, but will compare experimental measurements in section 6. The proportion of time spent in *SYNC* is given as T_{sync} and the proportion of time spent in *OFFDUTY* is given as $T_{offduty}$. $T_{scan} = \mu/\phi_{max}$ and $T_{offduty} = 1 - (\mu/\phi_{max})$.

5.2.1. Policy A: Null policy. Under the *null policy* nodes never enter the *OFFDUTY* state. The synchronisation window length is always $\mu = \phi_{max}$ such that the radio module is always in the *SYNC* state, except for assigned duty periods in which nodes temporarily assume the *ONDUTY* state. We use this policy as a baseline against which to compare the other policies as the resulting energy consumption is the upper boundary of all possible CDAP policies.

5.2.2. Policy B: Hyperbolic Decline policy. A counter is maintained of the number of consecutive successful synchronisation event predictions, σ. Initially, $\sigma = 0$. Every time a synchronisation event is detected within the appropriate synchronisation window σ is incremented; if the no event is detected in the window then the counter is reset as $\sigma = 0$. The window length is taken as $\mu = \phi_{max}$ until at least $\sigma = \chi$ consecutive successful predictions occur, after which $\mu = max\big(\phi_{max}/(\sigma + 1), 2\kappa\big)$. This allows the policy to rapidly shrink the synchronisation window but not until the system begins to stabilise. Increasing χ delays window shrinking for longer, reducing premature shrinking but also reducing potential energy savings.

5.2.3. Policy C: Moving Average Error policy. A buffer Ξ records the ξ most recent *phase prediction error* magnitudes, which are the unsigned magnitude of the difference between the predicted and measured phase for a phase neighbour synchronisation event. If the predicted event was not observed, a *null* value is recorded. Taking the average of the non-null members of Ξ provides a moving average prediction error, ϵ. If

no non-null members of Ξ exist we take $\epsilon = \phi_{max}$. A value $\nu \in [1, 2]$ is typical.

We set $\mu = max(\nu\epsilon, 2\kappa)$ during each system epoch. This defines the window size in terms of actual observed prediction errors; in essence the network nodes *learn* the local timing uncertainty and adapt dynamically. $\nu \geq 1$ is a scaling constant which determines the extent to which the next phase error can be bigger than recent historical phase errors and still allow nodes to reliably detect synchronisation events.

5.3. Measuring efficiency

We define the metrics Q as the rate at which a node consumes energy. The sensornet executes the protocol as runtime t increases in the interval $[0, \infty]$. Q measures the mean energy consumed per node per unit time during a measurement period $[t_{start}, t_{stop}]$. Unlike $P_0 - P_2$ (see section 4.5) these measurements apply to individual nodes rather than populations, so we measure Q for each node and take the mean to normalise metrics in cell population size n.

Q: Rate of energy consumption of a node. Measured in *Watts*. Defined in the range $(0, \infty)$. The ideal value of $Q = 0$.

As per metrics $P_1 - P_3$ defined in section 4.5 we observe that the measured values will differ if the measurement period considers the entire runtime of the network or is restricted to the equilibrium state. The former measures the system in transition from the initial state to the stable state, whereas the latter measures the long-term behaviour of the stable system. Both options are valid but non-equivalent, and both are measured in the experiments described in section 6.

6. Experimental results

We implemented CDAP in a modelled sensornet. We assess whether the empirical measurements match the theoretical predictions of sections 4 and 5.

6.1. Experimental configuration

We consider a set of homogeneous sensornets which are identical in all respects except for hardware platform. We use energy profile data for the MICA2 and MICAz motes extracted from manufacturer product data sheets [15] and two independent sets of experimentally measured energy profile data for the MICA2 mote [16], [17]. We label these energy profiles $E_1 - E_4$.

We assume that no application data packets traverse the network during the test period. The behaviour induced by CDAP is fully independent of any distributed or localised application running on the sensornet infrastructure. It is therefore unnecessary to model any sensornet application as it would have no impact on CDAP, and furthermore it is unhelpful to do so as this results in confounding of the CDAP and application influences on system energy profile.

A cell population of $n = 10$ is selected because this is an energy-efficient cluster size for typical 1000-node sensornets [18]. All experiment nodes are located in the same cell. All experiments begin with initial node phases in the same randomised distribution.

We set $\kappa = 0.01s$ as this an order of magnitude greater than the shortest complete packets for the selected mote platforms and as such offers a substantial safety margin. The exact value of the system epoch size e is irrelevant as we measure the passage of time in complete epochs so we set $e = 10s$ as this is orders of magnitude greater than κ. For the desynchronisation moving average parameters we set buffer size $p = 10$, required non-null proportion $q = 0.5$, and maximum consecutive nulls $r = 5$. There are no policy-specific parameters for policy A experiments. We select $\chi = 5$ for policy B experiments. We select window scaling factor $\nu = 1.5$ for policy C experiments.

In the experiments we measure the number of system epochs, j, required for each policy to reduce synchronisation window size to $\mu = 2\kappa$. We set duty period scaling factor $\eta = 1$ to evaluate worst-case duty period overlap. We measure the metrics $P_0 - P_2$ and Q for two periods; the first being the time from network start-up to CDAP during convergence, and the second being a longer duration after convergence.

6.2. State timing

Figure 2 shows synchronisation window shrinking against time under policies $A - C$. The response for successor and predecessor synchronisation events are very similar, but for clarity we display only the former. We see that Policies B and C significantly outperform policy A in minimising synchronisation window length and converge to $\mu = 2\kappa$, but perform identically until the algorithms are permitted to begin window shrinking. This happens at epoch $j = \chi$ for policy B, and at epoch $j = p$ for policy C.

Policy A is trivially converged at epoch 0, policy B reaches convergence at epoch 94, and policy C reaches convergence at epoch 23. Whereas policy B induces a smoother and more predictable window size decline, policy C generally offers a smaller window size after the respective algorithms are allowed to begin. This highlights the advantage conferred by policy C *learning* network characteristics as opposed to policy B *assuming* network characteristics, where these assumptions must be pessimistic to prevent unacceptable synchronisation prediction misses. More significantly, each prediction miss requires policy B

to restart at $\mu = \phi_{max}$ whereas policy C can tolerate some prediction misses before resetting $\mu = \phi_{max}$.

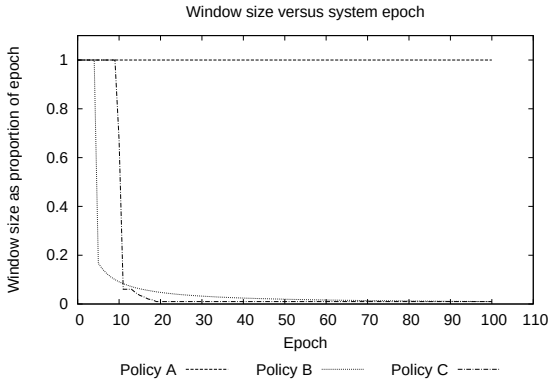

Figure 2. Window size

Note that Policies B and C confer significant efficiency improvements prior to convergence; this is simply the point at which the window size hits the predefined minimum threshold, preventing the algorithms from reducing window size further. If this threshold was not enforced the policies would shrink the synchronisation window length to zero as $j \to \infty$. This is ideal in a theoretical system in which synchronisation transmissions are instantaneous, but infeasible for real systems in which $\kappa \neq 0$.

Policy	Convgd.	SCAN	SYNC	ONDUTY	OFFDUTY
A	-	0.0000	0.9000	0.1001	0.0000
B	N	0.0378	0.0590	0.1056	0.7976
	Y	0.0000	0.0200	0.0999	0.8800
C	N	0.3462	0.1536	0.1244	0.3757
	Y	0.0000	0.0203	0.0999	0.8796

Table 1. Proportion of time in protocol states

Table 1 illustrates the proportion of time spent in each CDAP state (see section 4.3). We see that under all policies $A-C$ the measured proportions match the theoretically predicted proportions. The proportion of each epoch spent by nodes in *ONDUTY* is 0.1, which is the expected value of $1/n$. When each policy has converged the time spent in *SCAN* is 0, demonstrating that the reduced synchronisation window size is compatible with accurate synchronisation event observation.

For policy A the proportion spent in *SYNC* is 0.9 and the time spent in *OFFDUTY* is 0 as predicted. In contrast, under policies B and C the proportion in *SYNC* is 0.02, the minimum window size threshold 2κ, with the remaining 0.88 in *OFFDUTY*.

Prior to convergence, policies B and C display behaviour that is better than that of A but not as good as the converged behaviour. We conclude that Policies $A-C$ all assign duty periods of appropriate length, but Policies B and C can achieve this while assigning the majority of time to a low energy state.

6.3. Cell coverage

Table 2 illustrates the proportion of time in which 0, 1, or $\geq$2 nodes are in the *ONDUTY* state (see section 4.3). We see that under all policies $A-C$, P_0 and P_2 are very close to zero and P_1 is very close to 1 in the converged state, and hence are very close to the ideal values. Each policy $A-C$ is highly effective at maintaining mutual exclusion with exactly one node undertaking cell-wide duties at any given time.

Policy	Convgd.	P_0	P_1	P_2
A	-	0.0001	0.9948	0.0051
B	N	0.0001	0.9890	0.0109
	Y	0.0001	0.9948	0.0051
C	N	0.0100	0.9223	0.0677
	Y	0.0001	0.9945	0.0040

Table 2. Proportion of time for cell coverage

For policy C we see slightly poorer behaviour prior to reaching convergence. We attribute this to the relatively short convergence period of the window shrinking algorithm coinciding with the settling period of the underlying network; when convergence is attained the values of $P_0 - P_2$ are excellent.

6.4. Energy efficiency

Table 3 states values of Q, the mean energy consumption rate at each node (see section 5.3, for policies $A-C$. Taking policy A, the least energy efficient option, as a baseline for comparison, we observe that policies B and C offer lower energy consumption. This is true before and after CDAP reaches convergence, with both B and C offering similar energy efficiency.

Policy	Convgd.	E_1	E_2	E_3	E_4
A	-	0.0540	0.0831	0.0441	0.0470
B	N	0.0301	0.0360	0.0168	0.0193
	Y	0.0187	0.0252	0.0125	0.0148
C	N	0.0427	0.0609	0.0317	0.0345
	Y	0.0194	0.0275	0.0135	0.0159

Table 3. Mean energy consumption rates (Watts)

Under energy model E_1 there are improvements of 65% and 64% under policies B and C respectively. Under E_2 the improvements are 70% and 67%, under E_3 the improvements are 72% and 69%, and finally under E_4 the improvements are 69% and 66%. Although energy models $E_1 - E_4$ differ in composition,

we see a recurring qualitative outcome. Policies B and C offer significant improvement in energy efficiency over the baseline policy A, with policy B offering a slight advantage over policy C. Network designers must, however, balance this against the better reliability and shorter convergence time of policy C.

7. Conclusions

In section 3 a set of desired research objectives was defined, against which we now state our findings.

Obj 1: *Define a lightweight duty allocation protocol for unicellular sensornets such that exactly one node is active at any given time.*

The *Cyclic Duty Allocation Protocol* (CDAP) was defined in section 4 and extended to perform dynamic state management of mote subsystems in section 5. The protocol manages a unicellular sensornet such that exactly one mote is available for communication duties at any given time, and that the corresponding energy cost is shared equally by all participating motes.

Obj 2: *Evaluate the duty allocation protocol experimentally and theoretically to assess its efficacy and energy efficiency.*

Theoretical estimates of protocol performance and energy efficiency are defined in sections 4 and 5, against which empirical measurements are compared in section 6. It is shown that the protocol achieves its aims, constructing a near-optimal duty schedule with significantly improved energy efficiency.

Future work will consider potential interactions with application traffic flows to give further protection against clashes with synchronisation packets. Additional energy efficiency improvements are possible if the assumption that every cell must always have exactly one active node can be relaxed safely.

References

[1] M. Caccamo, L. Zhang, L. Sha, and G. Buttazzo, "An implicit prioritized access protocol for wireless sensor networks," in *Proc. Real-Time Systems Symposium*, 2002, pp. 39–48.

[2] V. Paruchuri, S. Basavaraju, A. Durresi, R. Kannan, and S. Iyengar, "Random asynchronous wakeup protocol for sensor networks," in *Proc. International Conference on Broadband Networks*, 2004, pp. 710–717.

[3] S. Ni, Y. Tseng, Y. Chen, and J. Sheu, "The broadcast storm problem in a mobile ad hoc network," in *Proc. IEEE International Conference on Mobile Computing and Networking*, 1999, pp. 151–162.

[4] Y. Gao, K. Wu, and F. Li, "Analysis on the redundancy of wireless sensor networks," in *Proc. International Conference on Wireless Sensor Networks and Applications*, 2003, pp. 108–114.

[5] F. Ye, G. Zhong, S. Lu, and L. Zhang, "PEAS: A robust energy conserving protocol for long-lived sensor networks," in *Proc. IEEE International Conference on Network Protocols*, 2002, pp. 200–201.

[6] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proc. Information Processing in Sensor Networks*, 2005, pp. 65–70.

[7] R. Mirollo and S. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM Journal Of Applied Mathematics*, vol. 50, no. 6, pp. 1645–1662, December 1990.

[8] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," in *Proc. International Conference on Mobile Computing and Networking*, 1999, pp. 263–270.

[9] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, "Instrumenting the world with wireless sensor networks," in *Proc. International Conference on Acoustics, Speech, and Signal Processing*, 2001, pp. 2033–2036.

[10] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is not sloth," in *Proc. USENIX 1995 Technical Conference*, 1995, pp. 17–17.

[11] C. Hua and T. Yum, "Asynchronous random sleeping for sensor networks," *ACM Transactions on Sensor Networks*, vol. 3, no. 3, 2007.

[12] K. Wu, Y. Gao, F. Li, and Y. Xiao, "Lightweight deployment-aware scheduling for wireless sensor networks," *Mobile Networks and Applications*, vol. 10, no. 6, pp. 837–852, 2005.

[13] Y. Pan and X. Lu, "Energy-efficient lifetime maximization and sleeping scheduling supporting data fusion and QoS in Multi-Sensornet," *Signal Processing*, vol. 87, no. 12, pp. 2949 – 2964, 2007.

[14] J. Degesys, I. Rose, A. Patel, and R. Nagpal, "DESYNC: self-organizing desynchronization and TDMA on wireless sensor networks," in *Proc. Information Processing in Sensor Networks*, 2007, pp. 11–20.

[15] Crossbow Technology Inc. product data sheets, http://www.xbow.com/Products/, accessed 09/01/2008.

[16] V. Shnayder, M. Hempstead, B. Chen, G. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proc. International Conference on Embedded Networked Sensor Systems*, 2004, pp. 188–200.

[17] H. Davis and R. Miller, "Power management for MICA2 motes," in *Proc. Southern Appalachian Symposium on Programming Languages and Systems*, October 2005.

[18] D. Wang, "An energy-efficient clusterhead assignment scheme for hierarchical wireless sensor networks," *International Journal of Wireless Information Networks*, vol. 15, no. 2, pp. 61–71, June 2008.