An Improved Lightweight Synchronisation Primitive For Sensornets

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

Abstract

Sensornets must allocate limited computation and energy resources efficiently to maximise utility and lifetime. This task is complicated by the need to coordinate activity between nodes as sensornets are necessarily real-time collaborative systems. In this paper we present and evaluate lightweight adaptive protocols based on pulse-coupled oscillators to synchronise tasks within a unicellular sensornet. A near-optimal schedule is constructed and dynamically maintained under non-ideal network conditions.

1 Introduction

Sensornets are bound to the physical environment into which they are deployed, implying real-time requirements on the sensornet and the distributed application it supports [7]. Sensornet application designers must establish when network nodes are available to send, process or receive messages. Reliability is important in achieving probabilistic guarantees of real-time behaviour, as lost messages may imply delays, missed deadlines and wasted energy.

In a general network we might coordinate distributed behaviour through distributed scheduling and routing protocols. This is typically difficult to achieve in a sensornet. Motes are generally equipped with the bare minimum resources required to support the distributed application. Lightweight protocols with probabilistic success measures are typically favoured over heavier but more reliable alternatives [6]. No global clock exists, with distributed scheduling rendered difficult by many independent local clocks steadily drifting out of synchronisation [12].

Consider a situation in which we have a finite set of nodes with each node capable of broadcasting in a shared wireless medium, located such that they form a fully-connected network cell [1]. These broadcasts can be received by any member of the cell which is listening to the medium, or by any nearby external entities. Communication may occur with the cell, or with neighbouring cells and base stations to exchange data and tasking messages.

Localised synchronisation protocols can support distributed applications that would otherwise fail without a global clock. We would like to arrange the timing of periodic events such that the delay between any two consecutive events is identical, and the ordering of events within each system epoch is identical. Under these conditions we can build protocols, such as the Cyclic Duty Allocation Protocol [14], upon this primitive which exhibit regular, predictable and fair allocation of duties. Long-term stability can be achieved despite imperfect clocks and connectivity.

This paper addresses the following problems which represent novel contributions. We implement a lightweight feedback-driven protocol to build and maintain a cyclic duty schedule based on a variant of the biologically-inspired *synchronisation* phenomenon [11]. We show that this protocol works well under ideal network conditions, and propose improved versions that also perform well under adverse network conditions. The protocol rejects timing error from clock drift and jitter. The protocol is scalable and effective in mobile networks where nodes join and leave cells unpredictably. The work makes no assumptions about the low-level communications mechanisms, is stateless with regard to knowledge shared between motes, and does not require a global clock or internode synchronisation of local clocks.

This paper is structured as follows. Section 2 places this work in the context of the relevant literature. Sections 3 to 8 examines the *desynchronisation* primitive and propose improved versions, with section 9 examining these experimentally. Section 10 summarises the findings of the work.

2 Related work

Many sensornet tasks and data flows are at least approximately periodic [1], typically as a consequence of periodic interaction with the physical environment. In this paper we consider distributed synchronisation protocols applied in closed finite systems of cooperating sensornet nodes. We wish to synchronise cyclic duty periods in strict round-robin order within a larger system period known as an *epoch*.

A rich and diverse body of literature exists on the

978-1-4244-5113-5/09/$25.00

scheduling of periodic tasks in general systems; a comprehensive survery can be found in [8]. The periodic nature of sensornets suggests a *cyclic schedule* rather than a *priority-driven* or *deadline-driven* approach. A *distributed* algorithm is necessary without a central controller to enforce shared schedules. *Dynamic* algorithms are required where motes are mobile or unreliable.

Caccamo *et al.* [1] propose a hybrid scheduling approach for multicellular sensornets. A *Frequency Division Multiplex* (FDM) strategy allocates different channels to adjacent cells by map colouring. Within each system-wide epoch an *Earliest Deadline First* (EDF) algorithm, distributed and replicated exactly at each node in a cell, allocates a proportion of equal-length frames to intra- and inter-cellular traffic. Traffic between adjacent cell pairs is managed under strict geographic *cyclic executive*.

PalChaudhuri *et al.* [12] define a protocol for clock synchronisation which is *adaptive* to the needs of a distributed application. It supports *relative synchronisation* where network nodes minimise the relative difference between local clocks, and *external synchronisation*. The overhead is relatively high; during each synchronisation iteration each node requires $O(n^2)$ bidirectional data packet exchange with all neighbours, and execution of a linear regression calculation. This cost is justified if the application requires nodes to collaborate at a *specific* time, rather than the lesser requirement that they collaborate at the *same* time.

Synchronisation and *desynchronisation* are biologically-inspired primitives in which a closed finite system of periodic oscillators converge to a steady equilibrium state. System level coordination is an emergent property of independent agents implementing simple rules. Under *synchronisation* all oscillators fire simultaneously in the steady state [11], whereas under *desynchronisation* the oscillator firing times are evenly distributed in time in the steady state [4].

Wang and Aspel [16] observe that these primitives converge rapidly without global clocks, adapting automatically to changing cell population. Unlike the *Phase-Locked Loop* (PLL) and *Delay-Locked Loop* (DLL) approaches, which offer similarly predictable and lightweight synchronisation behaviour, there is no requirement to maintain continuous contact between peers in the wireless medium.

A decentralised algorithm is defined by Lucarelli and Wang [9] in which a sensornet of arbitrary logical topology applies a variant of the synchronisation-seeking algorithm defined in [11]; it is not required that the network graph is fully connected. Each sensornet node acts as a periodic oscillator but propagates its synchronisation signal only to nodes that are one hop away in the network topology. Over time, the entire system converges on a *synchronised* state.

DESYNC-TDMA is a TDMA algorithm based on *desynchronisation* to *perfectly interleave periodic events to occur in a round-robin schedule* in a fully-connected network [4]. Each node acts as a periodic oscillator. Synchronisation signals are exchanged with peers defined by physical connectivity rather than logical network topology. The relative phase of signals measured within cyclical epochs is used to dynamically correct perceived error. Rapid convergence on a stable limit-cycle is guaranteed under ideal conditions, but disproportionately lengthy restabilisation periods result from small signal timing perturbations or network errors.

Christensen *et al.* [2] suggest that similar approaches can be applied in self-configuring systems of highly mobile robots. The physical topography of the implicit network can change very quickly owing to the high mobility of nodes. These self-organising strategies are particularly beneficial in highly dynamic and unpredictable situations, such as *Vehicular Ad-Hoc Networks*, where less agile approaches would struggle to maintain coordinated schedules.

3 The desynchronisation primitive

In this section we consider the elements of the *desynchronisation* primitive [4], and the properties of the converged equilibrium state. We use the standard definitions of pulse-coupled oscillator systems [11], but rephrase these from a global system viewpoint to a local node viewpoint as individual nodes do not have complete system knowledge.

3.1 Building blocks

Assume we have a set Σ of nodes $S_1 \cdots S_n$ where $n \geq 2$; if $n = 1$, there is obviously no need for internode 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 event V_i exactly once. 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. These events need not be related to any functionality of the sensornet application. However, if the application naturally produces periodic events of this type, perhaps as part of a distributed sensing function, then these application events can be reused for synchronisation.

Distributed protocols and applications can use the resulting stream of observeable synchronisation events, occurring every t time units, as the foundation for coordinated activity. Periodic application events required to occur with frequency $f = 1/t$ can be triggered directly by observed synchronisation events. Application events specified at other frequencies may use harmonics of the synchronisation frequency f; other arbitrary relationships can be supported.

Between observed events nodes must use a local clock. During this period it is possible that the local timer of each

node may drift by varying amounts, until the next observed event corrects the effects of this drift. However, it is reasonable to assume that commodity timers based on quartz crystals offer acceptably small and predictable drift between observed events [13] as typical drift rates are very small [3]. The impact of clock drift is evaluated in section 9.5.

3.2 Equilibrium state properties

For a *desynchronised* system in a stable state, the ordering of events V_i is stable from epoch E_j to E_{j+1} and the elapsed time between any two consecutive events is equal to e/n. A stable state conformant to these specification is known as an *equilibrium state*; as time is continuous, if there exists at least one equilibrium state there exists an infinite number of such states. Fortunately, all equilibrium states are equivalent and equally acceptable.

Before the system reaches the equilibrium state it is possible that the inter-event time can change; when the equilibrium state is reached, it can not. The specific stable ordering is unimportant, though it is a deterministic consequence of the initial state of the system and the set of shared rules, but the inter-event time t is always $t = e/n$.

Although we have defined that the period of all events V_i is equal such that $\forall i : p_i = e$, we do not explicitly define the offset o_i of each periodic event V_i within a stable epoch; this is a deterministic consequence of running a coordination protocol based on the *desynchronisation* primitive as described below. The order of offsets o_i within an epoch E_j defines the order of events V_{ij}, but any ordering offers equivalent coordination behaviour within the network cell.

As each epoch E_j is of equal length e, and each event V_i is periodic, for a given epoch we can define the *phase* of each event relative to the epoch start. If t_{ij} is the time from the start of epoch E_j to event V_{ij} then phase $\psi_{ij} = t_{ij}/e$. For a stable system the identity of the epoch is not relevant, so $\psi_i = t_i/e$. This gives phase measurements defined in the range $\psi \in [0, 1)$. Any value $\psi \notin [0, \psi_{max})$ is equivalent to $\psi \bmod \psi_{max}$ as a consequence of modular arithmetic inherent in phase calculations. Equivalent behaviour is observed if all values of ψ are scaled linearly with maximum phase ψ_{max} taking some arbitrary real value, so we will use ψ_{max} in the analysis but use the explicit $\psi_{max} = 1$ when presenting experimental results.

If we now consider the inter-event time t in terms of phase, we find that the phase difference between any two consecutive events V_x and V_y is $\Delta\psi = (e\psi_{max})/n$. To achieve this equal $\Delta\psi$ we must schedule the events V_i evenly in time throughout an epoch. This schedule must also ensure a margin of $\Delta\psi$ exists between the last event of the previous epoch and the first event of the given epoch, and between the last event of the given epoch and the first event of the following epoch. Within a given epoch the time before the first event and the time after the last event must sum to $\Delta\psi$ to provide sufficient margin.

Conversely, under *synchronisation* we would require that all periodic events occur simultaneously within each epoch. Whereas this would also be usable as the foundation for coordinated distributed activity, the duration between observeable synchronisation events would be n times longer than under *desynchronisation*. This would increase the risk of errors deriving from clock drift and other timing inaccuracies, but offers no saving in energy consumption or overhead.

3.3 Attaining equilibrium state

Sections 3.1 and 3.2 describe the system from the viewpoint of an external observer with access to the entire system. Now consider the viewpoint of a participating node S_i which can observe events occurring at other nodes but has no other information. Each node tracks the passage of time using its internal clock, corresponding to a local measure of phase ϕ_i in the range $[0, \phi_{max})$ where $\phi_{max} = \psi_{max}$ as given above. Each node S_i applies the algorithm independently, so we can define this algorithm using only locally-available data and assume that each participating node executes the same algorithm in parallel.

The difference between ψ and ϕ is that ψ gives a system-wide measure of the passage of time as measured in phase units, whereas ϕ_i gives the local measure of the passage of time as experienced by a single node S_i. This is significant because each node S_i does not have omniscient access to information available to any other node, and does not have access to any system-wide overview. As protocol designers we can use system-wide information to measure the effectiveness of a network design, but the nodes upon which the protocols are implemented have access only to information learned from their environment.

Consider an arbitrary epoch E_j. When $\phi_i = \phi_{max}$ the event V_i is triggered at node S_i and ϕ_i is reset to 0. Each node S_i is aware of the time at which its own event V_i executes, and the times at which the instantaneously preceding and following events $V_{i\beta}$ and $V_{i\gamma}$ occur. The node S_i does not know, and does not need to know, the identity of the other nodes $S_{i\beta}$ and $S_{i\gamma}$, the *phase neighbours* of S_i, at which $V_{i\beta}$ and $V_{i\gamma}$ occur respectively. However, S_i will influence and be influenced by its phase neighbours.

Assume a node S_i executes event V_i, and observes preceeding event $V_{i\beta}$ and succeeding event $V_{i\gamma}$ which may or may not occur in the same epoch E_j. Node S_i measures the duration $t_{i\beta}$ between $V_{i\beta}$ and V_i, and the duration $t_{i\gamma}$ between V_i and $V_{i\gamma}$, using its internal clock. We convert these timings into relative phase differences as $\phi_{i\beta} = -(t_{i\beta}/\phi_{max})$, and $\phi_{i\gamma} = (t_{i\gamma}/\phi_{max})$. Note that $\phi_{i\beta}$ is negative as the predecessor phase neighbour event $V_{i\beta}$ must occur before V_i, but is nevertheless equivalent to the positive value $(\phi_{i\beta} \bmod \phi_{max}) \in [0, \phi_{max})$.

In the equilibrium state described in section 3.2, all events V_i will be equidistant between preceeding event $V_{i\beta}$ and succeeding event $V_{i\gamma}$. We define *phase error* for node

S_i as $\theta_i = \phi_{i\beta} + \phi_{i\gamma}$, which is the phase amount by which the timing of event V_i differs from the desired stable state value. When a equilibrium state is attained, $\forall i : \theta_i = 0$.

The phase error for node S_i can also be found as $\theta_i = (t_{i\gamma} - t_{i\beta})/\phi_{max}$ by substituting the definitions of $\phi_{i\beta}$ and $\phi_{i\gamma}$ given above; this alternative notation is equivalent but may be easier to implement directly where nodes sleep for periods during which local phase ϕ_i is not monitored.

As soon as node S_i becomes aware of succeeding event $V_{i\gamma}$ during each epoch, node S_i can execute the phase adjustment procedure. Recall that node S_i has an internal clock which it uses to maintain a measure of local phase ϕ_i. Node S_i evaluates its phase error θ_i when succeeding event $V_{i\gamma}$ is observed. We now use θ_i to adjust ϕ_i by the *phase change* amount $\Delta\phi_i$, which will either enlarge or contract the duration until the next execution of event V_i. This is achieved by immediately setting $\forall i : \phi_{inew} = \phi_{iold} + \Delta\phi_i$. Note that this $+\Delta\phi_i$ adjustment must also be applied to any phase measurements of other events stored within node S_i.

We define $\Delta\phi_i = -f_\alpha\theta_i$ where $f_\alpha \in (0, 1]$ represents the *feedback proportion*. Higher f_α values give faster convergence but less stability, whereas lower f_α values give a system which takes longer to reach a equilibrium state but is more stable to the deleterious effects of noise.

This local phase correction directly changes the behaviour of node S_i and indirectly changes the behaviour of phase neighbours $S_{i\beta}$ and $S_{i\gamma}$; during the following epoch all events V_i will be closer to their equilibrium-state equilibrium phase ψ_i. Given an otherwise unchanging network, $\forall i : ||\Delta\phi_{ij}|| \rightarrow 0$ as $j \rightarrow \infty$ in successive epochs [11]. If $\theta_i = 0$ then the phase change $\Delta\phi_i = 0$ as well; no special action needs to be taken. Note that systems implementing this primitive may be sufficiently converged to support useful application work before reaching full convergence.

Algorithm 1 defines the primitive behaviour executing at each node $S_i \in \Sigma$ under the original version of the primitive. Variables not defined within the algorithm itself take the standard meanings used elsewhere in this document.

Algorithm 1 : Original primitive variant A at node S_i

Require: Observed predecessor sync phase, $\phi_{i\beta} = nil$
Require: Observed successor sync phase, $\phi_{i\gamma} = nil$
while monitoring local phase ϕ_i increasing over time **do**
  if sync event $\neq V_i$ observed **then**
    if $\phi_{i\gamma} = nil$ **then**
      $\phi_{i\gamma} \Leftarrow \phi_i$
      if $\phi_{i\beta} \neq nil$ **then**
        $\theta_i \Leftarrow \phi_{i\beta} + \phi_{i\gamma}$
        $\Delta\phi_i \Leftarrow -f_\alpha\theta_i$
        $\phi_i \Leftarrow (\phi_i + \Delta\phi_i) \bmod \phi_{max}$
        $\phi_{i\gamma} \Leftarrow (\phi_{i\gamma} + \Delta\phi_i) \bmod \phi_{max}$
      end if
    else
      $\phi_{i\gamma} \Leftarrow \phi_i$
    end if
  end if
  if $\phi_i \geq \phi_{max}$ **then**
    if $\phi_{i\beta} = nil$ **then**
      $\phi_{i\beta} \Leftarrow \phi_{i\gamma}$
    end if
    $\phi_{i\gamma} \Leftarrow nil$
    $\phi_i \Leftarrow 0$
    fire own sync event V_i
  end if
end while

4 Measuring solution quality

Recall from section 3.2 that upon reaching a equilibrium state the set of events has an even temporal distribution. For a given node S_i we know that when local phase $\phi_i = 0$ the event V_i is exactly equidistant from both V_β and V_γ, and we know that the relative phase difference between V_β and V_γ is given by $2(\phi_{max}/n)$. It is therefore possible to measure the observed behaviour against this defined ideal to obtain estimates of solution quality at any given instant.

Each node can calculate these metrics using locally available data, perhaps using these to moderate local application behaviour. Ideally, all nodes would have the ideal value of all metrics.

M_1**:** *Allocated timeslot length.* In the equilibrium state each node is allocated communication duty for an equal proportion of each epoch. The metric is calculated for each node S_i as $M_{1i} = t_{i\beta} + t_{i\gamma}$ and is measured in *seconds*. The ideal value is $M_1 = e/n$.

M_2**:** *Asymmetry.* In the equilibrium state each node broadcasts its synchronisation pulse exactly equidistant from those of its phase neighbours with perfect symmetry. The metric is calculated for each node S_i as $M_2 = ||t_{i\beta} - t_{i\gamma}||$ and is measured in *seconds*. The ideal value is $M_2 = 0$.

M_3**:** *Node population estimate.* In the equilibrium state each node has sufficient information to accurately estimate the cell population, and hence to decide whether it should participate. The metric is calculated for all nodes as $M_3 = [e/(t_{i\beta} + t_{i\gamma})]$, and is measured in *nodes*. The ideal value is $M_3 = n$.

5 Implementing the primitive

Networks in which this primitive is applied can be modelled as a fully connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of network nodes and $\mathcal{E}$ represents the set of possible pairwise communication exchanges. We assume signal propagation, though not packet propagation, is instantaneous in the wireless medium. We cannot assume an atomic publisher-subscriber model in non-ideal networks.

The events V_{ij} 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 $S_i \in (\Sigma \backslash S_\alpha)$. 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. Recipients will use the time at which the *pulse* is received, rather than information encoded into the signal itself.

Precisely how this *pulse* is implemented is irrelevant to the content of this paper, because any implementation which successfully distributes the single bit messages at the appropriate times would convey the same source information to the algorithm. However, a typical implementation would be the transmission of the shortest possible header-only packet achievable under a given network stack. 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. In an ideal system $\kappa = 0$ such that $M_1 - M_3$ approach their ideal values as system time $t \to \infty$. In a realistic non-ideal system $\kappa > 0$, so we expect M_1 and M_2 to converge within $\pm\kappa$. As M_3 is rounded to the nearest integer we would expect it to converge on the correct integer if κ is sufficiently small.

6 Tuning

There are three parameters of the *desynchronisation* primitive; the number of nodes, n, the system epoch length, e, and the feedback proportion, f_α. Achieving acceptable network performance requires the setting of appropriate values of n, e and f_α. Appropriateness is defined in application-dependent and -independent factors.

The hardware in the deployment network may affect the possible range of n. This is particularly important where nodes are mobile or fragile; applications should continue to perform correctly when a single node leaves the cell. Application requirements may specify a minimum and/or maximum number of nodes to give a probabilistic guarantee of coverage of the physical region covered by the sensornet cell. n can never be higher than the number of nodes deployed into the environment, and can never be lower than 1 for any non-degenerate case, but between these bounds the appropriate value of n is application dependent. We conclude that n is significant but not tunable.

The network designer is largely free to set f_α to any defined value to obtain a reasonable tradeoff between responsiveness and stability. We examine the effect of different f_α values in section 9.2. Usually f_α is set to a high value to achieve good responsiveness, shortening the time to attaining *equilibrium state*. However, non-ideal network conditions can lead to inaccurate, noisy or missing inter-node synchronisation data. Unfortunately, the *desynchronisation* algorithm will respond as quickly to noise as to accurate data, harming solution stability. Network designers can reduce f_α, reducing feedback and increasing systemic *damping*, to minimise this effect at the cost of reduced responsiveness to real system changes. A better solution is given by the improved protocol variants defined in section 7.

The behaviour of the primitive is independent of e; virtually any value might be selected provided that $e \geq n\kappa$ to allow all n synchronisation messages to be transmitted within each epoch. Within each epoch, the proportion of time consumed by synchronisation is given by $n\kappa/e$. Larger values of e assign a greater proportion $p = 1 - (n\kappa/e)$ of each epoch for application usage rather than synchronisation; $p \to 1$ as $e \to \infty$. As the number of epochs required for the system to reach the required level of convergence is independent of e, if e is large then so is the wall time implied by these epochs. In highly mobile networks it is therefore useful to keep e relatively small, but sufficiently large for application-specified tasks to complete. However, synchronisation messages are typically very small; even relatively small e values are orders of magnitude greater than κ, such that p is insignificant and convergence is fast.

7 Improved variants of the primitive

In section 6 we observe that tuning the f_α parameter to increase responsiveness to timing signals has the unwanted side effect of increasing responsiveness to timing errors. Setting low values of f_α damps the response of the *desynchronisation* primitive, improving resilience to transient errors and network conditions at the expense of responsiveness to real network changes. It may be difficult to achieve an acceptable compromise through this single point of influence. In this section we propose an alternative approach.

Recall from section 3 that each node S_i can disregard all observed synchronisation events other than the phase neighbours of its synchronisation event V_i, and that the sources of these phase neighbour events do not change between system epochs. Normally node S_i will use exactly one instance of the predecessor event $V_{i\beta}$ and the successor event $V_{i\gamma}$ in calculating $\Delta\phi_i$. These single instances are most recent observations, which will occur at $\phi_{i\beta} = \phi_{max}(e/2n) \bmod \phi_{max}$ and $\phi_{i\beta} = -\phi_{max}(e/2n) \bmod \phi_{max}$ respectively in the *equilibrium state* from the local viewpoint of node S_i.

Rather than use the most recently observed values of $\phi_{i\beta}$ and $\phi_{i\gamma}$, we propose that each node maintains a *moving average* over the most recent m complete epochs, stored in two queue buffers of size m at each node. Each queue is initially populated with *nil* values which do not contribute to the moving average. During each epoch the new value is pushed on the head of the appropriate queue, and the oldest value is popped off the end of the queue. If no phase neighbour events are observed in a given epoch, a *nil* value is pushed on the queue instead of a measurement. This is required for well-defined behaviour in the degenerate case where node movement temporarily implies $n = 1$.

For a queue containing ν non-*nil* values, the *fill ratio* $\pi = \nu/m$ increases in $[0, 1]$ as $\nu \to m$. The minimum *fill ratio* π_{min} required to calculate meaningful moving averages is

specified by the application designer; larger values imply a greater delay until noise rejection behaviours are active, but have more data with which to work and hence are less susceptible to the influence of outliers.

When the node S_i is required to amend its local phase, as per section 3.3, the relative phases $\phi_{i\beta}$ and $\phi_{i\gamma}$ of phase neighbour events are calculated as the arithmetic mean of the associated buffer of recent historical values if $\pi \geq \pi_{min}$; otherwise, we revert to the original strategy of using the most recent observations directly. The underlying primitive remains fundamentally unaltered in this improved algorithm and hence retains its convergence properties, but operates on higher-quality source data. The network designer must still set an appropriate value of f_α.

To improve responsiveness we use variants of the plain moving average that give greater weighting to more recent values, but can still operate effectively when the value for the current system epoch is undefined as a result of a lost pulse. Assume we label the non-null historical data values in each buffer as $x_1, \ldots, x_m$ where x_m is the most recent. We employ an *exponentially weighted moving average* in which the weighting w of historical data point x_y is given as $w_y = y^z$ where $z \in \mathbb{R}$ is the *scaling exponent*. If $z = 1$ then we have the plain moving average. If $z > 1$ then newer data are more significant, whereas if $z < 1$ then older data are more significant. Usually $z > 1$ will be selected to give higher priority to newer data.

Algorithm 2 defines the primitive behaviour executing at each node $S_i \in \Sigma$ for the improved primitive variants B and C. Function Π returns the fill ratio of a given buffer. Function avg returns the average of the filtered values stored in a given buffer, where the type of average is appropriate to the selected primitive variant. Other variables and functions not defined within the algorithm itself take the standard meanings used elsewhere in this document.

Algorithm 2 : Primitive variants $B - C$ at node S_i

Require: Most recent observed sync phase, $\phi_\alpha = nil$
Require: Predecessor sync phase queue buffer, $Q_{i\beta} = \varnothing$
Require: Successor sync phase queue buffer, $Q_{i\gamma} = \varnothing$
Require: Peer sync event counter, $c = 0$

```
1:  while monitoring local phase φ_i increasing over time do
2:    if sync event ≠ V_i observed then
3:      c ⇐ c + 1
4:      φ_α ⇐ φ_i
5:      if c = 1 then
6:        Q_iγ ⇐ Q_iγ ∪ {φ_α}
7:        if Π(Q_iβ) > π_min ∧ Π(Q_iγ) > π_min then
8:          φ_iβ ⇐ −(φ_max − avg(Q_iβ))
9:          φ_iγ ⇐ avg(Q_iγ)
10:         θ_i ⇐ φ_iβ + φ_iγ
11:         Δφ_i ⇐ −f_α θ_i
12:         φ_i ⇐ (φ_i + Δφ_i) mod φ_max
13:         φ_α ⇐ (φ_α + Δφ_i) mod φ_max
14:         for all q_iβ ∈ Q_iβ do
15:           q_iβ ⇐ (q_iβ + Δφ_i) mod φ_max
16:         end for
17:         for all q_iγ ∈ Q_iγ do
18:           q_iγ ⇐ (q_iγ + Δφ_i) mod φ_max
19:         end for
20:       end if
21:     end if
22:   end if
23:   if φ_i ≥ φ_max then
24:     Q_iβ ⇐ Q_iβ ∪ {φ_α}
25:     if c = 0 then
26:       φ_α = nil
27:       Q_iγ ⇐ Q_iγ ∪ {nil}
28:     end if
29:     φ_i ⇐ 0
30:     c ⇐ 0
31:     fire own sync event V_i
32:   end if
33: end while
```

8 Cost analysis

The plain version of the *desynchronisation* primitive defined in section 3 requires only two items of data to be stored. As the local phase ϕ_i increases from 0 to ψ_{max} for some given node S_i any number of pulse events might be observed, but only the first and last are retained. The first corresponds to the successor pulse event $V_{i\gamma}$, and the last corresponds to the predecessor pulse event $V_{i\beta}$, that surround the local pulse event V_i. We require storage space for exactly two such timing data, as each value will be overwritten with new data during each epoch. Therefore, the storage overhead is $O(1)$ in node population, n. The algorithmic complexity is also $O(1)$ in n because the algorithm requires a small fixed number of steps to be executed during each epoch; there are no loops or other recursive constructs. This low overhead is highly desirable in sensornet systems which have few resources to allocate.

Now consider the moving average variants defined in section 7. Storage and computation overheads remain $O(1)$ in node count as the algorithm continues to consider only the two phase neighbour nodes, irrespective of any number of other participating nodes which might be present. However, we must now consider the number of event observation timing values, m, which contribute to the moving average on each execution of the algorithm. Note that this applies only to the calculation of the effective phase of events $V_{i\beta}$ and $V_{i\gamma}$; the phase adjustment algorithm is unaffected.

There exist algorithms to calculate simple moving averages that are $O(1)$ in storage and computation overhead, and if these are employed it is obvious that the moving average offers significantly improved performance with minimal increased overhead. However, a general moving average algorithm may be worse than $O(1)$ but no worse than

$O(m)$ in storage and computation and overhead, the latter being observed if the algorithm must consider all m contributing data on each execution.

We observe that each execution of the algorithm at each node is guaranteed to terminate in $O(1)$ time. However, the algorithm is executed once at each node during each epoch, so in this sense the algorithm never terminates. This latter condition is essential if the algorithm is to remain responsive to changing network conditions; it is obvious that no algorithm could respond after terminating.

For systems expected to be deployed into highly predictable and rarely changing environments, non-terminating algorithms may not be the most efficient choice. However, sensornets are typically deployed in highly unpredictable and changeable environments, and mobile ad-hoc networks are characterised by continual change; the algorithms described in this paper are an appropriate choice. For moderately changing environments, these primitives can be executed until equilibrium is reached, then cyclically suspended for significant periods then executing for short periods. During suspended periods the extant event schedule can be reused without incurring overhead, with schedule repair and recalibration occurring during execution periods.

9 Experimental results

We model the Crossbow MICA2 mote in our experiments. We set $\kappa = 1 \times 10^{-3}s$ as the time required for a synchronisation pulse transmission-reception pair to complete, and hence take this as the threshold deviation from the ideal value of metrics M_1 and M_2 within which we consider a system *converged*. As metric M_3 is inherently rounded we require the measured value to exactly match the ideal value. Each metric is measured at all nodes $S_i \in \Sigma$. We count the elapsed time in system epochs from network initialisation to the point at which the mean, minimum and maximum values measured across the participating nodes all fall within the defined threshold.

Unless stated otherwise we use a fixed cell population $n = 10\ nodes$ because this is an energy-efficient cluster size for typical 1000-node sensornets [15]. We label the plain desynchronisation algorithm as A, the *basic moving average* variant as B, and the *exponentially weighted moving average* variant as C. We select epoch length $e = 10s$ so that epochs are large compared to k and long enough for realistic tasks to complete between synchronisation events in time e/n. We select feedback $f_\alpha = 0.9$ yielding similar fast convergence under all variants $A - C$ (see section 9.2).

For variants B and C we set buffer size $m = 10$ to ensure that sufficient captured synchronisation data contributes to moving averages to reject the effect of outliers and timing error, but does not contain unacceptably stale historical data which may no longer be representative of current network conditions. We set fill ratio $\pi_{min} = 0.5$ assuming that synchronisation timing data extracted from fewer than half of the system epochs may be unrepresentative, although the protocol would continue to function under this condition. For variant C we specify scaling exponent $z = 2$ such that newer data exert more influence than older data.

We do not claim that these parameteric values are optimal. Selecting the most appropriate values for a given specific network is an optimisation problem which is beyond the scope of this paper. However, these values are typical and illustrative, and we show that useful behaviour is observed over broad ranges of the defined parameters.

To model other hardware platforms substitute a different κ, and to model other networks different values of n, e and f_α can be used; the results are qualitatively equivalent but quantitatively different. Note that metrics M_1 and M_2 approach their κ convergence limits asymptotically; it is possible to achieve a looser but acceptable degree of convergence in significantly shorter time. Network designers must tradeoff solution quality against algorithm efficiency when specifying network requirements.

Section 9.1 models coordinated and uncoordinated network deployment scenarios. Section 9.2 models networks of differing cell size and responsiveness requirements. Section 9.3 models situations in which mobile nodes enter or leave the physical region covered by a network cell, suspend or wake in response to duty cycle management protocols, or leave the network owing to hardware failure. Section 9.4 models networks where malfunctioning hardware, environmental obstacles or deliberate sabotage disrupts inter-node communications. Section 9.5 models networks where malfunctioning hardware, poor application design or extreme ambient temperature induces local timing errors.

9.1 Cell initial configuration

We define *initial configuration* as the set of initial node phases relative to the start of the first system epoch. In *random* initial configurations these starting phases are randomly distributed in the interval $[0, \psi_{max})$. In *ideal case* initial configurations these starting phases are evenly distributed in time, identical to the *desynchronised equilibrium state*. In *worst case* initial configurations all starting phases are equal, identical to the *synchronised equilibrium state*.

We begin by illustrating convergence of metrics from a *random* initial configuration. Figure 1 shows the mean values of metrics $M_1 - M_3$ across all nodes, with all measured values normalised to the range $[0, 1]$. Metrics were sampled at the end of each of the first 100 system epochs under the original algorithm variant A. Similar plots are obtained for variants B and C but are omitted owing to lack of space.

All metrics $M_1 - M_3$ can be approximated by sequences of the form $f(j) = 1/j + c$ in epoch j where c is some constant. We observe that M_1 very quickly approaches its limiting value. As epoch j increases the value M_{1j} alternates between higher and lower than the limit $M_{1\infty}$ with the difference $||M_{1j} - M_{1\infty}||$ quickly becoming small. M_3

also approaches its limiting value $M_{3\infty}$ quickly, though not as quickly as M_1, with relatively large perturbations from the idealised hyperbolic form explained by the quantisation of individual measurements to integral values (see section 4). M_2 converges more slowly than M_1 or M_3 but declines smoothly and monotonically toward the limit $M_{2\infty}$.

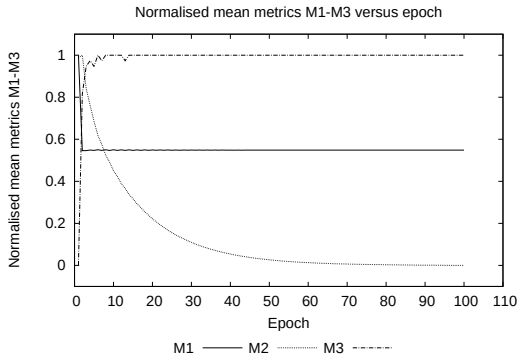

Figure 1. Normalised metrics for variant A

Table 1 presents the time required for metrics $M_1 - M_3$ to converge. Consider the behaviour when the system starts in the *best-case* configuration, equivalent to an *equilibrium state* of the algorithm. We see that the system maintains this ideal configuration for all metrics $M_1 - M_3$. This simply, but importantly, indicates that the algorithm will not take the system from an *equilibrium state* to a *non-equilibrium state*. We need not consider the *best-case* configuration further.

Now consider the M_1 metric. We see that M_1 reaches its converged value very quickly for all algorithm variants and all initial configurations. We conclude that all variants are highly capable in this regard under ideal network conditions and need not consider this metric further.

For all variants $A - C$, we see that all metrics $M_1 - M_3$ will converge in finite time starting from a *randomised* or *worst-case* initial configuration. In all experiments, reaching the convergence limit required more epochs from a *worst-case* initial configuration. This is unsurprising as the *worst-case* configuration is further from the *best-case* configuration than almost every *randomised* configuration, excepting *randomised* configurations that are also *worst-case*.

The number of epochs required to reach the convergence limit $M_{3\infty}$ is nearly the same for each algorithm variant $A - C$. This is a consequence of the calculation of M_3 rounding intermediate values to the nearest integral value, an effect which will dominate small variation in pre-rounded intermediate values as these converge.

Now consider the M_2 metric, which in all cases is the slowest to reach the convergence limit and therefore defines the point at which cells reach an *equilibrium state*. Starting with a *randomised* initial state we observe the epochs required for convergence is of the same order of magnitude for each algorithm variant, but convergence is reached somewhat faster under variant A than B or C; a smaller difference exists between values for variants B and C. This is explained by hysteresis effects; variant B calculates new values using historical data and variant A does not, so the output of variant B lags behind that of A. Variant C is somewhere between A and B both in the influence of historical data and the corresponding measured responsiveness.

We conclude that all algorithm variants $A - C$ are effective under ideal network conditions.

Initial state	Algorithm variant	Epochs to convergence			
		M_1	M_2	M_3	*MAX*
Random	A	3	25	11	*25*
	B	3	38	21	*38*
	C	3	37	21	*37*
Best	A	1	1	1	*1*
	B	1	1	1	*1*
	C	1	1	1	*1*
Worst	A	3	35	21	*35*
	B	3	56	24	*56*
	C	3	54	24	*54*

Table 1. Convergence times for metrics

9.2 Cell composition

In this section we measure the epochs required for all metrics $M_1 - M_3$ to converge to an *equilibrium state*. Figure 2 illustrates the relationship between f_α and the number of system epochs, y, which must elapse before the system reaches an *equilibrium state* under algorithm variants A and B; the trace for variant C is very similar to that of B and is omitted for clarity. Each value of f_α was evaluated with an identical *worst-case* initial configuration

Traces A and B are similar, though not identical, for $f_\alpha \in (0, f_{critical})$ where $f_{critical} \approx 0.91$. Up to this point, both A and B describe approximately hyperbolic traces such that the relationship between f_α and epoch of *equilibrium state* can be approximated by the form $f(j) = 1/j + c$ in epoch j where c is some constant. A difference in behaviour is noted for $f_\alpha > f_{critical}$; trace B continues its original hyperbolic path, whereas trace A grows quickly with $f_\alpha \in [f_{critical}, 1]$. Two distinct effects must be considered to understand this relationship.

In each epoch each node S_i amends its local phase by $\Delta\phi_i = -f_\alpha\theta_i$ where θ_i is the perceived phase difference between the local synchronisation event at $\phi_i = \psi_{max}$ and the midpoint of the phase neighbour events. The greater the value of f_α, the greater the proportion of perceived difference that is fed back into the system, pushing the system toward the *equilibrium state* more quickly. This explains the shape of trace B for $f_\alpha \in [0, 1]$, and the shape of trace A for $f_\alpha \in [0, f_{critical}]$.

Now consider trace A for $f_\alpha \in [f_{critical}, 1]$. θ_i is continuously variable but κ, the time for a pairwise exchange of synchronisation event, is constant. Converting κ from time units to phase units, the magnitude $||\Delta\phi_i||$ becomes small compared to the magnitude $||\kappa\phi_{max}||$. As the magnitude $||\kappa\phi_{max}||$ defines the uncertainty of the phase neighbour event midpoint measurement, it follows that the magnitude of the measurement error becomes significant compared to the magnitude $||\Delta\phi_i||$. This causes convergence to slow as the limit is approached. Each iteration of the procedure must attempt to correct for previous measurement errors within the new phase difference measurement.

If f_α is small, the proportion of this measurement error fed back into the system is also small, so its effect is insignificant. As f_α grows so does the proportion of measurement error feedback. Under variant B the measurement error is found in all stored samples. Although the error values are not explicitly available, as they derive from consecutive system epochs they are likely to be of similar magnitude, and they are as likely to be positive as to be negative. Taking the average of the samples will approximately cancel the measurement errors, so the effect of these errors does not become dominant. Under variant A there is no such cancellation effect, hence the effect of these errors becomes dominant. Defining convergence limits of significantly larger magnitude than κ would hide this phenomenon without actually addressing the underlying issue.

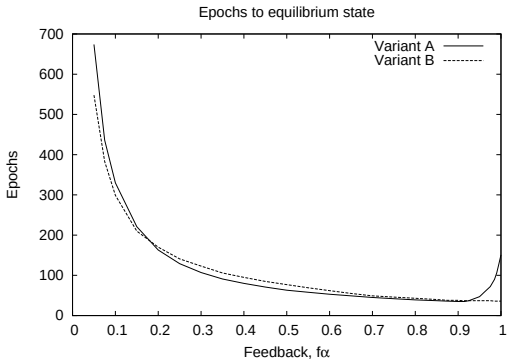

Figure 2. f_α versus epochs to equilibrium

Figure 3 illustrates the relationship between n and the number of system epochs, y, which must elapse before the system reaches an *equilibrium state* under algorithm variant A. Similar plots are observed for variants B and C but are omitted for brevity. As the cell n increases the general trend is that y increases too. It is notable that this increase is not monotonic, and does not conform readily to any well-known relationship. Despite the guarantee that the system will converge [11] it is difficult to predict the time required. This is a consequence of algorithm variants $A - C$ defining feedback-driven systems, in which the relationship between input and output is deterministic yet difficult to predict [5].

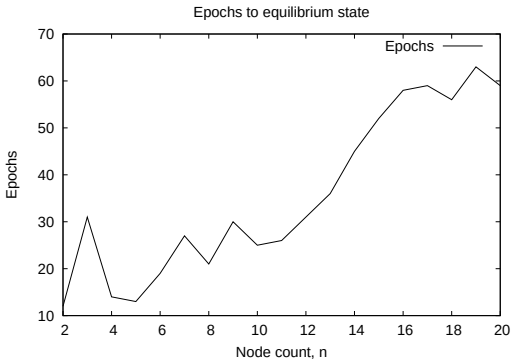

Figure 3. n versus epochs to equilibrium

Larger networks may be divided into multiple cells. Separate instances of the primitive operating in adjacent cells may interact if the communication range of some nodes extends beyond their own cell, if nodes cannot determine the cell from which a given transmission originates. If not managed, these interactions could cause disruption, similar to the *phantom pulse* effect examined in section 9.4.

These interactions can be exploited for beneficial effect. Extensions based on *entrainment* have been implemented which progressively synchronise equivalent transmissions in adjacent cells. This enables intercellular co-operation, mitigates the risk of clashing behaviour, and enables efficient handover of mobile nodes between cells. However, a detailed description is beyond the scope of this paper.

9.3 Cell population change

In this section we consider algorithm performance for cells starting in a stable *best-case* where a node is either added or removed from the cell population. We then measure the time required to reach a new *equilibrium state* where all metrics $M_1 - M_3$ are converged. We plot metric M_2 against epoch as this is the slowest to converge.

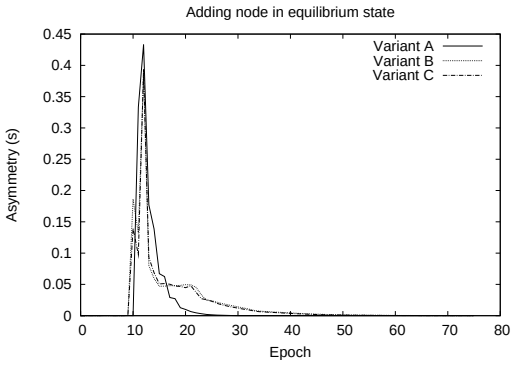

Figure 4. Adding node to stable system

Figure 4 shows a node being added to a stable 5-node system. Variant A requires 21 epochs to re-establish the

equilibrium state, variant B requires 58 epochs, and variant C requires 57 epochs. Figure 5 shows a node being removed from a stable 5-node system. Variant A requires 16 epochs to re-establish the *equilibrium state*, variant B requires 47 epochs, and variant C requires 46 epochs. The node removal experiments re-establish the *equilibrium state* more quickly because the new stable system is smaller than the new stable system in the node addition experiments.

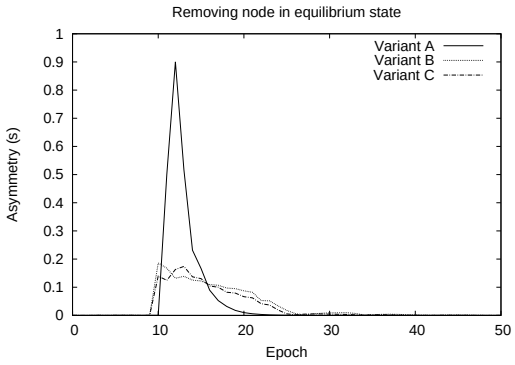

Figure 5. Removing node from stable system

In all cases the *equilibrium state* is re-established in finite time. Note that decreasing e reduces this time linearly. As the algorithm is capable of restabilising the cell schedule when a single node is added or removed, it is capable of dealing with multiple additions or removals as these can be decomposed into an equivalent temporally ordered sequence of single additions and removals. This is particularly helpful in networks of highly mobile nodes, in which cell membership is expected to change frequently.

9.4 Radio error resilience

In this section we consider algorithm performance for cells starting in a stable *best-case* where network conditions are non-ideal. It is possible that a synchronisation pulse transmission V_{ij} may fail to be heard at one or more of the intended recipients; we call each instance a *lost pulse*. Reception will either succeed or fail independently and atomically at each potential recipient. We measure performance where reception of an arbitrary pulse at an arbitrary node fails stochastically with probability $p \in [0, 1]$.

In figure 6 we set $p = 0.05$. For each variant $A - C$ exactly the same synchronisation pulse transmitter-receiver pairs were lost. We see that variants $B - C$ significantly outperform the original variant A significantly, with much smaller deviation in metric M_2 from the ideal value of $M_2 = 0$. Although neither variant B nor C cope perfectly with pulse loss, and there is little to pick between them, they offer substantially improved performance and stability.

Synchronisation pulses have minimal length and content; a *phantom pulse* is feasible where radio noise or corrupted packets are interpreted as a synchronisation pulse. We measure performance where nodes observe *phantom pulses* distributed randomly in time with rate r given in s^{-1}.

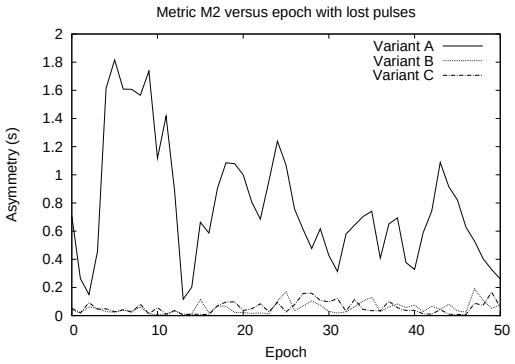

Figure 6. Lost pulses

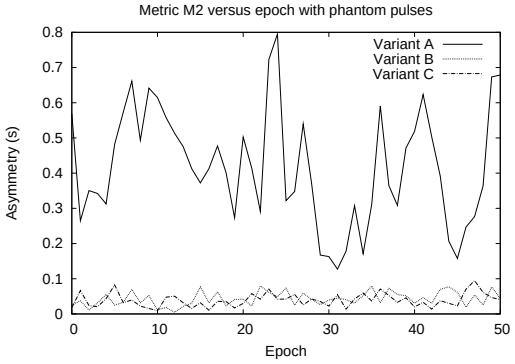

Figure 7. Phantom pulses

In figure 7 we set $r = 0.1\ s^{-1}$. For each variant $A - C$ exactly the same phantom pulses were heard by nodes. Again, we observe that variants $B - C$ offer significantly better stability and performance than variant A.

9.5 Clock error resilience

In this section we consider algorithm performance for cells starting in a stable *best-case* where timings are not accurate. *Jitter* in synchronisation pulse transmission times may result from non-ideal task scheduling algorithms or preemption by higher priority tasks at the sender node. Although many definitions are possible [8] we define the jitter ι of a given synchronisation pulse as the difference between the intended and actual transmission times, where ι is distributed normally as $\iota \sim N(\mu_\iota, \sigma_\eta^2)$. Transmission jitter affects both phase neighbours of the transmitter node, whereas individual radio error affect only a single receiver.

In figure 8 we set $\mu_\iota = 0s$, as early transmission is as likely as late transmission, and $\sigma_\eta = 0.1s$. For each variant $A - C$ pulse transmission times are subject to exactly the same jitter. We observe that variants $B - C$ show significantly better stability and performance than variant A.

Under variants $B - C$ the uncorrected error component is of the same order of magnitude as jitter standard deviation.

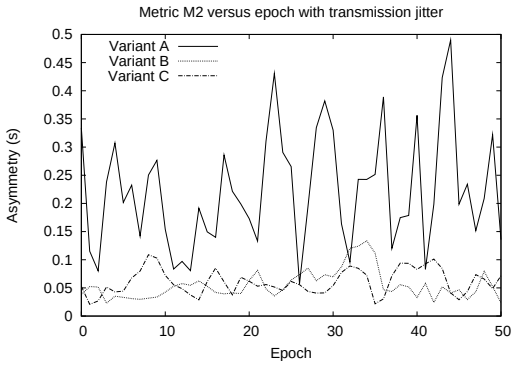

Figure 8. Jitter

Clock drift is observed if local node clocks are imperfect. As one second passes in the physical world the clock may measure more or less than one second passing, governed by a scaling factor $\eta > 0$. Perfect clocks have $\eta = 1$; manufacturing imperfections and variation between calibration and operational temperature tend to give $\eta \neq 1$ [10]. We assume each node clock has constant η [13], distributed normally as $\eta \sim N(\mu_\eta, \sigma_\eta^2)$. We set $\mu_\eta = 1$ to model clocks equally likely to run fast as to run slow. We set $\sigma_\eta = 1 \times 10^{-3}$, modelling drift rates with standard deviation several orders of magnitude greater than the 1×10^{-6} *seconds per second* drift typical of commodity quartz crystal timers [3]. Figure 9 shows variants $A-C$ perform acceptably in rejecting drift effects, with uncorrected error of the same order of magnitude as the drift. For variants $B - C$ we see some initial stabilisation as drift-laden measurements fill the buffers.

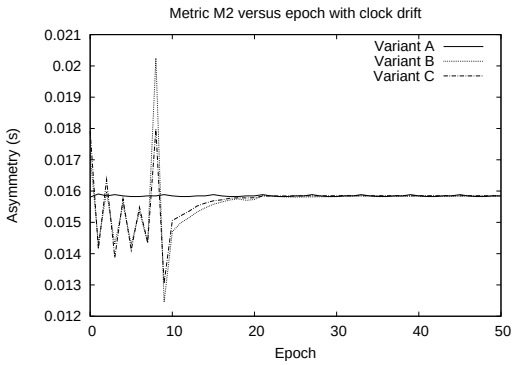

Figure 9. Clock drift

10 Conclusions

The *desynchronisation* primitive is lightweight and effective in coordinating activity within unicellular sensornets. However, the original version is prone to instability arising under common non-ideal timing and network conditions. We defined improved primitives yielding significant and measurable improvements in stability without sacrificing performance. Algorithmic and storage overheads are of the same order, $O(1)$, in cell population n as the original.

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*, pages 39–48, 2002.

[2] D. Christensen, D. Brandt, U. Schultz, and K. Stoy. Neighbor detection and crosstalk elimination in self-reconfigurable robots. In *Proc. Robot Communication and Coordination*, pages 1–8, 2007.

[3] F. Cristian, H. Aghili, and R. Strong. Clock synchronization in the presence of omission and performance failures, and processor joins. In *Proc. IEEE International Symposium on Fault-tolerant Computing Systems*, July 1986.

[4] 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*, pages 11–20, April 2007.

[5] R. Devaney. *A first course in chaotic dynamical systems*. Addison-Wesley, Reading, MA, 1992.

[6] 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*, pages 2033–2036, 2001.

[7] T. He, J. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *Proc. IEEE International Conference on Distributed Computing Systems*, pages 46–55, 2003.

[8] J. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, 2000.

[9] D. Lucarelli and I. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *Proc. SenSys'04*, pages 62–68, 2004.

[10] D. Mills. Precision synchronization of computer network clocks. *Computer Communication Review*, 24:28–43, 1994.

[11] R. Mirollo and S. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal Of Applied Mathematics*, 50(6):1645–1662, December 1990.

[12] S. PalChaudhuri, A. Saha, and D. Johnson. Adaptive clock synchronization in sensor networks. In *Proc. Information Processing in Sensor Networks*, pages 340–348, April 2004.

[13] M. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *Proc. IEEE Wireless Communications and Networking Conference*, volume 2, pages 1266–1273, March 2003.

[14] J. Tate and I. Bate. Energy efficient duty allocation protocols for wireless sensor networks. In *Proc. IEEE Conference on Engineering of Complex Computer Systems*, pages 58–67, June 2009.

[15] D. Wang. An energy-efficient clusterhead assignment scheme for hierarchical wireless sensor networks. *International Journal of Wireless Information Networks*, 15(2):61–71, June 2008.

[16] X. Wang and A. Apsel. Pulse coupled oscillator synchronization for low power UWB wireless transceivers. In *Proc. 50th Midwest Symposium on Circuits and Systems*, pages 1524–1527, August 2007.