# A Spatial Source Location Privacy-Aware Duty Cycle for Internet of Things Sensor Networks

MATTHEW BRADBURY, University of Warwick, UK
ARSHAD JHUMKA, University of Warwick, UK
CARSTEN MAPLE, University of Warwick, UK

Source Location Privacy (SLP) is an important property for monitoring assets in privacy-critical sensor network and Internet of Things applications. Many SLP-aware routing techniques exist, with most striking a trade-off between SLP and other key metrics such as energy (due to battery power). Typically, the number of messages sent has been used as a proxy for the energy consumed. Existing work (for SLP against a local attacker) does not consider the impact of sleeping via duty cycling to reduce the energy cost of an SLP-aware routing protocol. Therefore, two main challenges exist: (i) how to achieve a low duty cycle without loss of control messages that configure the SLP protocol and (ii) how to achieve high SLP without requiring a long time spent awake. In this paper, we present a novel formalisation of a duty cycling protocol as a transformation process. Using derived transformation rules, we present the *first* duty cycling protocol for an SLP-aware routing protocol for a *local eavesdropping attacker*. Simulation results on grids demonstrate a duty cycle of 10%, while only increasing the capture ratio of the source by 3 percentage points, and testbed experiments on FlockLab demonstrate an 80% reduction in the average current draw.

CCS Concepts: • **Networks** → **Sensor networks**; **Network privacy and anonymity**; *Network measurement*.

Additional Key Words and Phrases: Source Location Privacy, Wireless Sensor Networks, Duty Cycle, Fake Sources

**ACM Reference Format:**
Matthew Bradbury, Arshad Jhumka, and Carsten Maple. 2020. A Spatial Source Location Privacy-Aware Duty Cycle for Internet of Things Sensor Networks. *ACM Trans. Internet Things* 1, 1 (October 2020), 31 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Large deployments of wireless sensor networks (WSNs) as a component of the Internet of Things (IoT) are becoming increasingly utilised for applications such as asset monitoring and tracking [1, 31]. In an asset monitoring application, when the asset is detected a message is transmitted from the asset-detecting node (known as a *source* node) back to a base station (referred to as the *sink*). Since the communication range of the sensor nodes is typically less than the geographical distance between the source and sink, the message is routed over multiple hops to reach the sink.

To protect the *content* of this data, messages can be encrypted, however, the act of routing the message to the sink reveals *context* information about the event that encryption does not protect. One example of context information that needs to be protected is the location of the source, as revealing this information would allow an attacker to follow messages through the network to the source and capture the asset. The SLP problem was originally presented as the panda hunter

Authors' addresses: Matthew Bradbury, University of Warwick, Department of Computer Science, Coventry, CV4 7AL, UK, M.Bradbury@warwick.ac.uk; Arshad Jhumka, University of Warwick, Department of Computer Science, Coventry, CV4 7AL, UK, H.A.Jhumka@warwick.ac.uk; Carsten Maple, University of Warwick, WMG, Coventry, CV4 7AL, UK, CM@warwick.ac.uk.

game [25, 39], where pandas are being monitored by a sensor network and information is being reported to conservationists working at a base station via wireless nodes in the network.

Much work has been undertaken developing new routing strategies that provide SLP since the seminal work [12]. How effective these strategies are at protecting the location of the source against an attacker is the most important SLP metric. However, the energy consumption of the protocol is another crucial metric which needs to be minimised, as sensor nodes are typically powered by constrained power supplies (such as by 2 AA batteries). The number of messages sent is often used as a proxy for the energy usage of the protocol. This is useful as sending and receiving messages can individually be the most energy expensive tasks performed by a sensor node [31]. However, this does not factor in the energy cost of leaving the radio and CPU in an idle state while listening for messages. To save energy, sensor nodes tend to spend the majority of their lifetime asleep [19] and only periodically wakeup, a technique known as duty cycling.

The duty cycle of a node is defined as the percentage of time it spends awake. An *efficient* duty cycle protocol will have a sensor node sleep as long as possible. While a node is asleep it cannot receive messages, meaning there is a need for a *reliable* wakeup strategy to ensure nodes are ready to receive messages. This issue is crucial for SLP, as the time at which messages are sent and received (and if they are received) can affect the performance of an SLP-aware routing protocol as will be demonstrated in this work. Therefore, care needs to be taken to ensure messages are not lost due to the receiving node being asleep and that there is minimal delay in forwarding a message.

This paper proposes a duty cycle technique to reduce the energy cost of spatial SLP protocols, whilst ensuring a high level of SLP is provided against a local attacker. To achieve this, we present a novel formalisation of the duty cycling process, where duty cycling is considered as a transformation of one schedule into another. This enables the transformation technique to be applicable to a wider range of SLP routing protocols. To illustrate the duty cycle transformation process, *DynamicSPR* (previously presented in [6]) is used as a case study. We use a concise model of a schedule for non-duty cycled *DynamicSPR* and then apply a transformation to switch the state of the radio (on/off), based on the model, to obtain *DynamicSPR-S*.

Experiments via simulations and a real-world deployment show that a wakeup strategy based on this timing analysis can lead to large reductions in the time nodes are awake and performing idle listening. A duty cycle of 50% is achievable (where nodes spend 50% of their time sleeping) in cases in which 1 message is sent every second. Longer source periods produce lower duty cycles, with 10% achievable when 1 message is sent every 8 seconds. Experimental results from the FlockLab testbed show that the lower duty cycles lead to a reduced energy cost and make *DynamicSPR-S* more practical for deployment.

We make the following contributions in this paper:

(1) We propose a novel formalisation of duty cycling as a transformation process from one schedule to another.
(2) For a class of SLP-aware routing protocols, known as *spatial* routing protocols, we propose transformation rules to implement the transformation process.
(3) We model the schedule generated by a state-of-the-art non-duty cycled SLP-aware routing protocol, known as *DynamicSPR*.
(4) We present *DynamicSPR-S*, the duty-cycled version of *DynamicSPR* obtained after transforming it using the identified rules. The rules ensure that nodes only require local knowledge, thereby ensuring scalability.
(5) Simulations using COOJA and experimental deployments on real hardware via the FlockLab testbed demonstrate that a low duty cycle can be achieved whilst retaining high SLP levels on grid networks, non-grid networks, and non-grid networks with a hole. A comparison

against TinyOS's LPL show that the duty cycle obtain from the developed transformation rules performs better.

(6) A comparison against a modified Phantom Routing shows that arbitrary duty cycling will lead to poor SLP performance in other types of spatial SLP routing protocols.

The remainder of the paper is structured as follows. In Section 2 we discuss the related work before presenting a the models and a formalisation of the problem in Section 3. We then model the schedule generated by *DynamicSPR* in Section 4, and present the duty cycle algorithm in Section 5. Section 6 contains the experimental setup for the results in Section 7. We discuss implications in Section 8 and finally conclude with a summary in Section 9.

## 2 RELATED WORK

### 2.1 Source Location Privacy

The SLP problem first appeared around 2004 in the seminal work of [25, 39] in which the definition of the *panda-hunter game* was proposed. In the problem definition, an attacker who is physically present in the network attempts to locate valuable assets (pandas) by abusing a WSN deployed to monitor them. As the WSN periodically sends messages about the states of a panda to a base station, the attacker eavesdrops messages from nearby nodes and uses this information to identify the direction of the proximate source of a message. Using this information, they trace back through the network hop-by-hop to find the ultimate source of the message. Flooding was demonstrated to provide no privacy, as the attacker traces messages along the shortest path to find the source.

*2.1.1 Routing-based Solutions.* Phantom routing was proposed in the seminal work where the technique first sends messages along a directed random walk of a given length to a phantom node. After reaching the phantom node, the message is routed to the sink either via flooding [39] or by single-path routing [25]. The aim of phantom routing is to lead the attacker to the phantom source first, thus delaying the attacker on its way to the source. Privacy in this work was evaluated by an increase in the safety period, which is the time that it took for the adversary to capture the source.

Many solutions have aimed to improve upon the phantom routing strategy, such as choosing the next node based on angles between key nodes [32, 45], routing messages in a ring around the sink [17, 48, 51], and grouping messages to reduce the number of moves an attacker can make [5]. Alternatively, messages can be handed to a data mule to avoid revealing context information through broadcasts [41]. Of these solutions only [5] describes the use of retransmissions after failing to receive an acknowledgement to ensure reliability. Retransmissions are typically required when duty cycling, and in doing so the time at which a message is received is delayed, which will lead to different attacker behaviour.

*2.1.2 Fake Source-based Solutions.* An alternative technique initially proposed in the seminal work was to use fake sources that broadcast identical messages to the real source(s). Fake sources provide SLP by generating ⟨fake⟩ messages that lead the attacker away from the real sources [24]. Recent work has focused on dynamically determining parameter values online [6]. Fake source techniques are often criticised for their high energy usage, an issue we will address in this paper.

*2.1.3 Hybrid Routing/Fake Source Solutions.* Many techniques have since combined routing and fake sources to improve the levels of SLP provided. One example is tree-based diversionary routing [30], which imposes a tree structure on the network and then routes fake messages through the tree. PEM [47] extended the routing path from source to sink with branches of fake sources. The idea of fogs or clouds [13, 49] involves routing a message around a group of nodes called a fog and then onwards to other fogs before being forwarded to the source. Fog routing takes advantage of fake messages to provide additional privacy. In [21] the authors calculated where to allocate fake

sources and for how long to broadcast fake messages based on the probability that an attacker is at that potential fake source location. An energy model is used to evaluate this technique, but it focuses on the cost to send and receive data and does not consider the impact of a node sleeping.

*2.1.4 Other Solutions.* Other techniques to provide SLP against a local attacker, include using space in MAC beacon frames to disseminate messages from the source to other areas in the network before being routed to the sink [44]. Alternatively it is possible to allocate slots in a TDMA Data Aggregation Schedule in such a way that SLP is provided [27]. This TDMA-based technique did not investigate duty cycling the nodes and they always remained on. However, as TDMA is an effective way to achieve a low duty cycle, it is likely to lead to a low power consumption once implemented.

*2.1.5 Multiple Sources.* Few techniques have investigated protecting multiple sources simultaneously due to the complexities in provisioning temporal or spatial redundancy for multiple sources. The Red Paths protocol [3] avoided using previously used nodes in a path. However, the technique has a high capture ratio on small (>60%) and large (>20%) networks with multiple sources.

*2.1.6 Adversary With Global Visibility.* Techniques against a global attacker, often involve periodic broadcasting of all nodes in the network [34, 50] or via traffic decorrelation [40]. These techniques typically have an obvious duty cycle associated with them, but do not always lead to a low duty cycle. For example, [52] splits the network into a grid and then assigns half the nodes to one colour and half to another, a 50% duty cycle is obtained by alternating which coloured nodes should be on and which should be off.

*2.1.7 SLP Energy Models.* Many works adopt an energy model based on the cost to send and receive messages (e.g., [18, 30, 36]), but does not consider the impact of sleep states. Therefore, an issue with existing SLP protocols that provide privacy against a local attacker is that no duty cycle protocol is considered, which means that the energy cost of running these protocols is high since they do not put nodes to sleep. Furthermore, some of these techniques do not specify a retransmission strategy, meaning the applicability of techniques in popular operating systems such as ContikiMAC [14] and TinyOS's Low Power Listening [35] is uncertain. Using duty cycling may also lead to changes in the timing of messages or the path these messages take, so it is important to evaluate the impact a duty cycle protocol has on the ability of an SLP-aware routing protocol to continue protecting SLP. This means there is a need for evaluation of the impact of a duty cycle protocol, or for new duty cycle protocols for techniques that provide SLP against a local eavesdropping attacker.

### 2.2 Duty Cycling in WSNs

Many approaches have been developed to duty cycle wireless sensor nodes [11, 43]. These approaches are typically categorised based upon how nodes synchronise their wakeups and transmissions, either via (i) global time synchronisation, (ii) synchronising time across a subset of nodes (semi-synchronous), (iii) or not attempting to synchronise time (asynchronous). Approaches such as TDMA provide a straightforward way to assign slots of global time in which nodes are allowed to broadcast. This slot assignment can also be used to determine when a node should sleep. Existing work has already been performed to allocate a TDMA DAS schedule that provides SLP by assigning slots in a specific way [26]. Alternatively nodes could be clustered, using techniques such as LEACH [20], in order to synchronise time among a cluster of nodes. The cluster heads then interact with each other asynchronously.

Due to the cost of synchronising time, the majority of duty cycling techniques are asynchronous. TinyOS's LPL [35] performs periodic sampling for a preamble, if one is detected then another node is preparing to send a message and the radio should remain on to receive the message after the preamble. Alternatively, the receiver can periodically broadcast a beacon to indicate it is willing to

| Name | Symbol | Description |
|---|---|---|
| Local Wakeup | $t_s$ | How long the radio sleeps for after a wakeup. |
| Remote Wakeup | $t_{tx}$ | How long a packet's retransmission will be attempted for. |
| Delay After Receive | $t_d = 100$ ms | Keep the radio on for this period after receiving a message. |
| Maximum CCAs | $cca = 400$ | The maximum number of CCAs that a receiving node performs. |
| Minimum Samples Before Detect | $msbd = 3$ | The minimum number of detection while performing CCA before a packet is detected. |
| ACK Wait Delay | $t_{ack} =$ 256 jiffies | The amount of time a sender waits for an acknowledgement packet. |

Table 1. Parameters to TinyOS' DefaultLPL component

receive a message [46]. One of the main downsides to both of these approaches is the high latency introduced. Since the radio only needs to be duty cycled to address the high cost of idle listening, if another radio has a low energy cost to perform idle listening, then it can be used to detect wakeup signals [4]. The problem is that this type of technique requires non-standard hardware that sensor nodes are not usually equipped with. Another technique is Chase [10], which focuses on duty-cycling for flooding messages when nodes are broadcasting concurrently by dynamically extended the radio on time to improve delivery. Finally, if a WSN is sufficiently dense, then nodes can randomly wakeup and it will be likely that a node will be awake when a message is being sent.

As TinyOS is being used to develop these protocols, its duty cycle approach called Low Power Listening (LPL) will be used as a comparison. There are multiple stages to LPL; initially when the radio is listening for messages, a number of Clear Channel Assessment (CCA) checks are performed. CCA checks aim to detect a high energy signal being emitted by the radio on the sending node. Multiple checks are needed to ensure that a message is being received and to avoid staying awake when noise is detected. Once a certain number CCAs have been detected then the radio switches to receive mode. Once the radio is done with receiving then it will turn off for a period. When sending a message, the radio will be turned on and packets will be retransmitted for a period specified at compile time. Once an acknowledgement is received the radio has the option to turn off immediately. TinyOS has a number of parameters for its LPL as shown in Table 1. However, there have been several proposals to change a number of these parameters [22, 42].

A problem with these duty cycle approaches is that they require the routing protocols to retransmit messages if an acknowledgement is not received. However, this is an issue for SLP-aware routing protocols since they typically require that messages are sent at specific times. The delay introduced by long sequences of retransmissions causes messages to be broadcasted at different times than scheduled, so typical duty cycle techniques are unsuitable. Therefore, in order to reduce the energy cost of using *DynamicSPR*, a duty cycle technique will be developed based on extending the timing analysis from [6]. If nodes can predict when their neighbours will send messages, they can schedule their wakeups accordingly.

## 3 SYSTEM MODELS AND PROBLEM STATEMENT

In this section, we first describe the models assumed in this work and subsequently define the duty cycling transformation problem. We use the same models as in [6] in order to later facilitate a direct comparison with this previous work.

### 3.1 Network Model

A wireless sensor node (or node, for short) is a computing device that has limited computational capabilities and is equipped with a wireless radio to facilitate communication. A node also contains a number of sensors to gather data from its immediate environment. A WSN is a set of wireless sensor nodes with ad-hoc communication links between pairs of nodes. Each node in the network has a unique identifier. The nodes that are able to directly communicate with a node $n$ are called the neighbours of $n$ and this set of nodes is called the neighbourhood of $n$.

There exists one distinguished node in the network called a *sink*, which is responsible for collecting data and which acts as a bridge between the WSN and external networks. The nodes continuously sense for the occurrence(s) of predefined events, such as temperature exceeding a certain threshold. In this paper, (normal) nodes sense the presence of an asset, such as the presence of an endangered wildlife species, and then route the data via ⟨normal⟩ messages along a computed route (using some routing algorithm) to the sink for collection. Any node can be a data source and we assume there to be a single data source at any time. We assume that the network is *event-triggered*, i.e., when a node senses an event, it starts sending messages periodically to the sink for a certain amount of time. The duty cycling scheme we present in this paper will be operational once this event is triggered.

We assume two types of messages in the network: (i) control and (ii) application (or ⟨normal⟩) messages. We assume ⟨normal⟩ messages to be encrypted and that the source node includes its ID in the encrypted messages. Using the ID, the sink can infer an asset's location as we assume that the locations at which nodes are deployed by network administrators will be recorded. We do *not* assume that WSN nodes have access to GNSS in order to provide geolocation or timing information due to the resulting increase in energy cost. Control messages are messages used for network maintenance.

### 3.2 Safety Period Model

Given a long enough time, a solution for the attacker is to perform an exhaustive search of the WSN. To rule out such trivial solution, a metric called *safety period* was introduced in [25] to measure the privacy provided by SLP routing protocols. Safety period was defined as the number of messages that needed to be sent (or equivalently the time taken) before the attacker captured the source. A higher safety period indicated a higher level of privacy. However, this definition of safety period allows for an attack to take unlimited time to capture an asset, which is impractical, as the event may cease to exist during this search time. This means that the SLP problem can only be considered when it is *time-bounded*, i.e., the asset has to be captured within a certain time window. Therefore, we use an alternate, but analogous definition of *safety period*, where if the attacker fails to capture the source within this time period, it is considered to have failed to capture the source. How the safety period is calculated and what values are used will be described in Section 6.

### 3.3 Attacker Model

In [2] a classification of attacker strength for WSNs was defined using a combination of two dimensions: *presence* and *actions*. Presence captures the location and/or the radio range of the attacker, while actions capture the kinds of attacks the attacker can undertake. For example, presence could be local, distributed, or global; while actions could include eavesdropping and reprogramming. The attacker we assume is a *local distributed eavesdropper* based on the patient adversary, introduced in [25]. The attacker is classified as *distributed* because it has local visibility but is also mobile, thus gathering information at various nodes. Such an attacker is reactive in nature and it is specified using a start and an end condition and rules for movement, which are as follows:

(1) *Start*: The attacker initially starts at the sink.
(2) *End*: Once the source has been found, the attacker will cease moving.
(3) *Move rules*: When the attacker is co-located at a node $n$ and eavesdrops a ⟨normal⟩ message that it has not received before from a neighbour $m$, the attacker moves to $m$. Thus, in a normal setting, the attacker is geared to moving closer to the source as it only follows unique messages.

Previous work (such as [25]) has assumed that the attacker has the ability to identify whether a message has been previously responded to. We also make this assumption and implement it by having the attacker record and compare the message type and sequence number. If this assumption is not made, then any routing protocol may lead the attacker away from the source. The attacker will respond to both ⟨normal⟩ and ⟨fake⟩ messages, as ⟨fake⟩ messages are encrypted and padded to be indistinguishable from ⟨normal⟩ messages. The attacker will ignore any encrypted control messages, such as network health maintenance, neighbour detection, etc.

We assume that the attacker has the capability to perfectly detect which direction a message arrived from, that it has the same radio range as the nodes in the network, and also has a large amount of memory to keep track of information such as messages that have been heard. This is commensurate with the attacker models used in [6, 24, 25].

In this work, the attacker starts at the sink because the sink is the one location in the network where the attacker is *guaranteed* to eavesdrop a message from the source node, irrespective of the routing protocol used. The attacker could potentially start at any location in the network, however, the attacker may not receive messages due to their location not being on the route from the source to the sink. We assume that the sink is located at a base known to the attacker such as a military base or a field station used by scientists monitoring wildlife.

*3.3.1 Capabilities.* This attacker model is classified on the bottom of the total order of attacker capabilities (shown in Figure 1). This is because some stronger attacks would weaken the attacker's ability to capture the source by leaking information to the WSN regarding the attacker's position. For example, if the attacker attempted to disrupt the functioning of the network (e.g., by jamming transmissions), the attack would reduce the quantity of useful information that could be gathered. If an attacker attempted to broadcast messages to influence the routing protocol or to collaborate with other attackers, then the WSN could potentially detect an intrusion attack and respond by ceasing to broadcast around the attacker (similar to [37]).

$$\textit{eavesdrop} \rightarrow \textit{crash} \rightarrow \textit{disturbing} \rightarrow \textit{limited passive} \rightarrow \textit{passive} \rightarrow \textit{reprogramming}$$

Fig. 1. Attacker capability hierarchy proposed in [2].

Performing certain attacks such as breaking into a sensor node to obtain encryption keys (i.e., passive attacks) *are* good strategies for an attacker trying to defeat SLP. The problem with such an attack is that it is also time consuming. In [2, p. 11] it was predicted that a key stealing attack takes around 30 minutes to perform in the field (not counting preparation time elsewhere or the time it takes to find, obtain, and open a sensor). As our solution to SLP aims to provide a high level of SLP within a specific safety period, if the time taken to obtain encryption keys is larger than the safety period then the attacker will have failed to capture the source within this safety period. Then the attacker would have achieved better results by simply eavesdropping. Thus, in the context of SLP in WSNs, one of the most powerful type of local attackers that can be had is the distributed eavesdropper which we assume in this work.

**Algorithm 1** Attacker interaction with Duty Cycle

▷ A schedule sched is a sequence of activities (node_id, activity, time) where activity is one of $\{\text{on}, \text{off}, \text{rcv}(\mathcal{M}), \text{send}(\mathcal{M}, r)\}$
captured $\leftarrow 0$
$\mathbb{A} \leftarrow$ Sink ▷ Attacker's start position
1: **for** s $\in$ sched **do**
2: **switch** s **do**
3: **case** $(i, \text{recv}(\mathcal{M}), t)$ **do pass**
4: **case** $(i, \text{on}, t)$ **do pass**
5: **case** $(i, \text{off}, t)$ **do pass**
▷ If attacker receives a unique ⟨normal⟩ message from a neighbour, then it moves there.
6: **case** $(i, \text{send}(\mathcal{M}, r), t)$ **do**
7: **if** $\mathbb{A} \in$ Neighbour$(i)$ **then**
8: **if** ShouldMove$(\mathbb{A}, i, \mathcal{M}, r, t)$ **then** ▷ As defined by the rules in Section 3.3
9: $\mathbb{A} \leftarrow i$
10: **if** $i$ = src **then** ▷ If the attacker has reached the source, then stop
11: captured $\leftarrow 1$
12: **return**

### 3.4 Problem Statement Formalisation

There are two main classes of SLP-aware routing protocols: spatial and temporal [23]. The objective of a spatial SLP-aware routing protocol is to ensure that the attacker takes a path which is longer than the shortest path to reach the asset. Conversely, the objective of temporal SLP-aware routing protocols is to delay the attacker on its way to the asset, even if it takes the shortest path. The problem we address in this paper focuses on spatial routing protocols. Typically, such protocols make use of fake sources to lure the attacker onto longer paths to the source. Thus, there will be ⟨normal⟩, ⟨fake⟩, and control messages in the network.

Given a network topology $G = (V, E)$, the location of a source, the location of the sink, we model the specification of an SLP routing protocol as a set of *schedules*. A schedule is a sequence of *events*, where an event is a 3-tuple (node_id, activity, time). In a non-duty cycled SLP routing protocol, there are two types of activities to be considered:

(1) Sending a message: $\text{send}(\mathcal{M}, r)$.
(2) Receiving a message: $\text{rcv}(\mathcal{M})$.

A message can be of the following types: ⟨normal⟩, ⟨fake⟩, or control. $\text{send}(\mathcal{M}, r)$ specifies the type of message $\mathcal{M}$ to be sent while $r$ captures whether the transmission is a unicast or broadcast. If it is a unicast, then $r$ is a node id, while nothing is specified if it is a broadcast. Thus, the schedule generated by a non duty cycled SLP routing protocol is an interleaving of send and receive events.

### 3.5 Duty Cycling Modelling

Having modelled an SLP routing protocol as a schedule, we view a duty cycled SLP-aware routing protocol as a transformed schedule. The schedule is a sequence of annotated activities called events such as the radio or CPU being powered on when a message needs to be received or sent. Since the duty cycled SLP protocol is the result of a transformation, we now consider four activities, to include two new ones:

(1) Sending a message: $\text{send}(\mathcal{M}, r)$.
(2) Receiving a message: $\text{rcv}(\mathcal{M})$.
(3) Switching on: on. Which only switches on when the radio was off.

| Symbol | Description |
|---|---|
| $\mathcal{N}, \mathcal{C}, \mathcal{F}$ | ⟨Normal⟩, ⟨Choose⟩ and ⟨Fake⟩ messages ($\mathcal{M}$ is used as a placeholder these three types) |
| $S_i(\mathcal{M})$ | The time at which node $i$ sends message $\mathcal{M}$ |
| $R_i(\mathcal{M})$ | The time at which node $i$ receives message $\mathcal{M}$ |
| $\alpha$ | The time it takes a message to travel one hop |
| $\Delta(i, j)$ | The distance in hops between nodes $i$ and $j$ |
| $P_{src}$ | The Source Broadcast Period (the difference between the time two sequential ⟨normal⟩ messages are sent from the source) |
| $EW(\mathcal{M})$ | Early Wakeup for message $\mathcal{M}$ |
| $LS(\mathcal{M})$ | Late Sleep for message $\mathcal{M}$ |
| TFS | Temporary Fake Source |
| PFS | Permanent Fake Source |
| TailFS | Tail Fake Source |
| $\#_{\mathcal{F}}(j)$ | The number of ⟨fake⟩ messages sent by TFS $j$ |
| $D_{TFS}(j)$ | The duration of a TFS $j$ |
| $P_{TFS}(j)$ | The period at which TFS $j$ sends ⟨fake⟩ messages |
| $P_{PFS}(j)$ | The period at which PFS $j$ sends ⟨fake⟩ messages |
| $I_{TFS}(j)$ | The initial delay between $j$ becoming a TFS and it sending its first ⟨fake⟩ message |
| $\tau_{TFS}(j)$ | The time the node $j$ becomes a TFS |

Table 2. List of Symbols and Acronyms

(4) Switching off: off. Which only switches off if the radio is not in use.

An SLP protocol with a 100% duty cycle will have a sequence of on activities at the head of the schedule and a sequence of off events at the end of the schedule, i.e., $\forall n \in V \cdot (n, \text{on}, 0)$ appears as a subsequence at the start of the schedule and $\forall n \in V \cdot (n, \text{off}, \infty)$ appears as a tail of the sequence. There will then be an interleaving of send and rcv activities in between these two subsequences.

Then, using this model, we view the design of a duty cycled SLP-aware routing protocol as the transformation of a (non-duty cycled SLP routing) protocol to a (duty cycled SLP routing) protocol, whereby the transformation process inserts *on* and *off* events into the schedule at specific times. As such, the movement of an eavesdropping attacker in the presence of a duty cycled schedule is given in Algorithm 1. The attacker only updates its position when a node in its neighbourhood is sending a message (lines 9–15) else they dismiss any other event in the schedule. How an attacker decides to move (line 8) is implemented using the rules specified in Section 3.3.

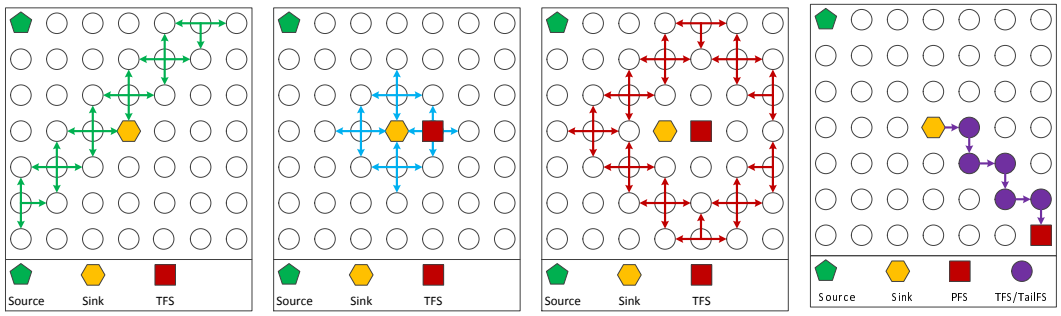
Overall, we consider a duty cycle $DC$, to be a function $DC : SCHEDULE \rightarrow SCHEDULE$. In the next section, we present simple rules to transform the original protocol in a duty cycled protocol. The focus on using simple rules is to ensure that the rules do not negatively impact on the efficiency of the non-duty cycled SLP-aware protocol.

## 4 TRANSFORMATION RULES AND *DYNAMICSPR*

In this section we propose a set of rules for the transformation of a spatial non-duty cycled SLP-aware routing protocol to its corresponding duty cycled protocol. We will subsequently show how they are applied to the *DynamicSPR* routing protocol [6] a non-trivial state-of-the-art non-duty cycled SLP-aware routing protocol to obtain *DynamicSPR-S* a duty cycled version of the protocol.

### 4.1 Transformation Rules

A spatial SLP-aware routing protocol defines a schedule for message transmissions. We denote the original schedule by $\mathcal{S}$, which is a sequence of activities where, for all nodes, *on* starts at time 0




(a) The source floods the network with ⟨normal⟩ messages, repeating every $P_{src}$ seconds.

(b) After the sink receives the first ⟨normal⟩ message it sends a ⟨choose⟩ message to start the fake source allocation.

(c) The ⟨fake⟩ message flood lures the attacker away from the source.

(d) The *DynamicSPR* fake source spread in a directed random walk away from the sink over time.

Fig. 2. Initial steps and fake source spread of the *DynamicSPR* algorithm [6].

and off starts at time $\infty$, i.e., they begin in an on state and never switch off. Using $\mathcal{S}$, we obtain $\mathcal{S}'$, the duty cycled schedule, as follows:

(1) When a node $n$ sends a message $\mathcal{M}$, at time $t$ in $\mathcal{S}$, then $n$ sends $\mathcal{M}$ at $t$ in $\mathcal{S}'$ and then switches off at $t + \alpha$, i.e., $(n, \text{send}(\mathcal{M}, r), t) \in S \Rightarrow ((n, \text{on}, t - \epsilon) \in S \wedge (n, \text{send}(\mathcal{M}, r), t) \in S' \wedge (n, \text{off}, t + \alpha) \in S')$.
(2) In order for a node $n$ to receive a message $\mathcal{M}$, at time $t$ in $\mathcal{S}$, then $n$ will need to switch on at $t - EW(\mathcal{M})$ and off at $t + LS(\mathcal{M})$ if no other component is using the radio. The early wakeup $EW(\mathcal{M})$ and late sleep $LS(\mathcal{M})$ capture the difference between the time at which a message is expected to be delivered and when it is delivered. Specifically, $(n, \text{rcv}(\mathcal{M}), t) \in \mathcal{S} \Rightarrow ((n, \text{on}, t - EW(\mathcal{M})) \in \mathcal{S}' \wedge (n, \text{off}, t + LS(\mathcal{M})) \in \mathcal{S}' \wedge (n, \text{rcv}(\mathcal{M}), t) \in \mathcal{S}')$.

Executing these translation rules is linear in time with respect to the number of events in the schedule $\mathcal{S}$, hence $O(|\mathcal{S}|)$. The complexity of this transformation can be related to the network $G = (V, E)$ on which the routing protocol is deployed as follows: Denote by $\theta$, a routing protocol-dependent maximum number of send and receive events that occurs on a node and by $|V|$ the number of nodes in the network. Then the complexity of the transformation can also be expressed as $O(\theta|V|)$ as $|\mathcal{S}| \propto \theta|V|$. This is because, for each event on each node, the complexity to transform that event into a duty cycled event is $O(1)$ and the cardinality of $\mathcal{S}$ equals the sum of the number of events for each node $n \in V$.

### 4.2 *DynamicSPR* Summary

In this section we present a brief summary of the *DynamicSPR* algorithm introduced in [6]. The algorithm works as follows:

(1) Initially the sink node sends messages to establish certain network knowledge (such as each node's sink distance).
(2) When an asset is detected, the source node repeatedly sends a ⟨normal⟩ message $\mathcal{N}_i$ with a time period between messages of $P_{src}$, beginning with $\mathcal{N}_1$.
(3) When the sink receives $\mathcal{N}_1$ it unicasts a ⟨choose⟩ message $\mathcal{C}$ to the neighbour it wants to become a fake source.
(4) When a node receives $\mathcal{C}$ it becomes a temporary fake source (TFS), or permanent fake source (PFS) if the node believes itself to be the furthest node from the sink.

(5) A TFS broadcasts a ⟨fake⟩ message $\mathcal{F}_i$ with period $P_{TFS}$ for a duration of $D_{TFS}$. When the duration expires it unicasts a ⟨choose⟩ message $C$ to the next fake source and also becomes a tail fake source (TailFS), that sends ⟨fake⟩ messages until a further fake source is detected.
(6) A PFS broadcasts a ⟨fake⟩ message $\mathcal{F}_i$ with period $P_{PFS}$.

The aim of *DynamicSPR* is to have the fake sources perform a directed random walk away from the sink and source to a location that is far from both nodes as shown in Figure 2d. ⟨Fake⟩ messages are used to lure the attacker to the location of the fake source instead of the real source.

### 4.3 Schedule Model for *DynamicSPR*

In this section we present a schedule model generated by *DynamicSPR*. We will then present the transformation for *DynamicSPR*, making it duty cycled *DynamicSPR-S*.

Existing asynchronous duty cycling techniques are unsuitable for *DynamicSPR* (as will be demonstrated with experiments using TinyOS LPL in Section 7). Therefore an alternate technique needs to be developed to duty cycle the nodes. To achieve this, the timing analysis performed in [6] will be extended to develop a schedule for when the radio should be turned on and off. There are two components that need to be implemented, one that focuses on the ⟨Normal⟩ messages that contain data from the source and another that focuses on the protection provided with ⟨Fake⟩ and ⟨Choose⟩ messages (which are protocol messages that control the spread of fake sources).

An important caveat (of this implementation) is that the timings need to be deterministic. This means that some previous definitions of *DynamicSPR* that were left unspecified, now need to be defined. Further, other parameters that involve a random component cannot be used. For example, the number of ⟨fake⟩ messages to be sent under the Rnd strategy (where either 1 or 2 messages are randomly sent per the TFS duration), cannot be used.

A benefit of this timing analysis is that it occurs locally on each node. This means that there is no need for time synchronisation across the network and no need for extra protocol messages to disseminate timing information. It is important that this scheme does not add much computation overhead, otherwise the expected broadcast times will not match the actual broadcast times which will cause the duty cycle to fall out of synchronisation.

### 4.4 ⟨Normal⟩ Message Timings

This section details when a node should wakeup and sleep to ensure delivery of ⟨normal⟩ messages. The source *src* (or *s*) periodically sends the $i^{\text{th}}$ ⟨Normal⟩ message at $S_{src}(\mathcal{N}_i)$.

$$S_{src}(\mathcal{N}_i) = iP_{src} \tag{1}$$

A node $j$ will receive $\mathcal{N}_i$ at the time $R_j(\mathcal{N}_i)$, where $\Delta(src, j)$ is the distance between the source and $j$ in hops, and $\alpha$ is the time a message takes to be broadcasted at each hop.

$$R_j(\mathcal{N}_i) = S_{src}(\mathcal{N}_i) + \alpha\Delta(src, j) \tag{2}$$

The time between a node $j$ receiving $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ is $\hat{R}_j(\mathcal{N}_i)$:

$$\hat{R}_j(\mathcal{N}_i) = R_j(\mathcal{N}_{i+1}) - R_j(\mathcal{N}_i) = P_{src} \tag{3}$$

Due to unreliability in the network, messages will not always arrive precisely at this time. Figure 7a shows the spread in the amount of time it takes a ⟨normal⟩ message to travel from the source to the sink. In some cases there can be up to 120 ms difference in the time it takes a node to receive a message. Therefore, it is important to wake up early $EW(\mathcal{N})$ and sleep later $LS(\mathcal{N})$ around the expected delivery time of the next ⟨normal⟩ message. The times that the node $j$ is asleep and awake

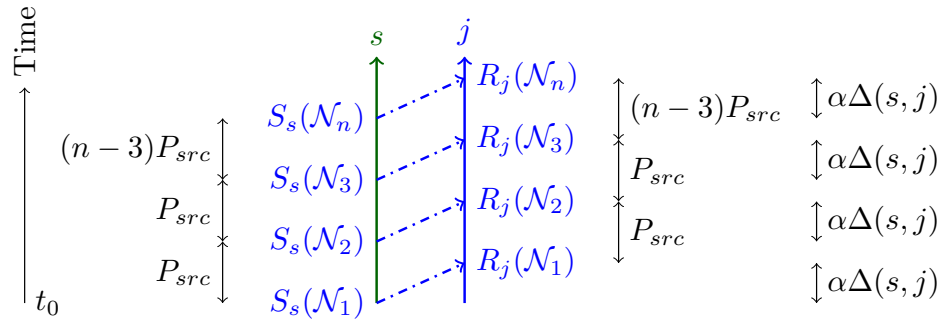



Fig. 3. Timing of events for the source $s$ sending ⟨Normal⟩ messages $\mathcal{N}_n$ and a node $j$ receiving them. With ⟨normal⟩ messages represented by dot-dashed arrows.

(starting from boot at time 0) are defined by:

$$awake(\mathcal{N}_1) = \begin{bmatrix} \textbf{from} & 0 \\ \textbf{to} & R_j(\mathcal{N}_1) + LS(\mathcal{N}) \end{bmatrix} \tag{4}$$

$$awake(\mathcal{N}_i) = \begin{bmatrix} \textbf{from} & \textbf{end}(asleep(\mathcal{N}_{i-1})) \\ \textbf{to} & \textbf{end}(asleep(\mathcal{N}_{i-1})) + EW(\mathcal{N}) + LS(\mathcal{N})) \end{bmatrix} \tag{5}$$

$$asleep(\mathcal{N}_i) = \begin{bmatrix} \textbf{from} & \textbf{end}(awake(\mathcal{N}_i)) \\ \textbf{to} & \textbf{end}(awake(\mathcal{N}_i)) + \hat{R}_j(\mathcal{N}_i) - EW(\mathcal{N}) - LS(\mathcal{N})) \end{bmatrix} \tag{6}$$

A diagram of the timing of events when sending ⟨normal⟩ messages is shown in Figure 3. The source node sends ⟨normal⟩ messages to the normal node $j$ every $P_{src}$ which is represented by the dot-dashed arrows. Node $j$ receives a ⟨normal⟩ message $\alpha\Delta(s,j)$ time units after it is sent, since this is the expected amount of time it takes messages to travel through the network along the shortest path. Once node $j$ receives a ⟨normal⟩ message $m$ at $R_j(\mathcal{N}_m)$ it knows that the next ⟨normal⟩ message $m+1$ will arrive at $R_j(\mathcal{N}_m) + P_{src}$ and during this time interval node $j$ can sleep.

### 4.5 ⟨Fake⟩ and ⟨Choose⟩ Timings

For ⟨fake⟩ and ⟨choose⟩ messages, there is additional complexity to consider when compared to ⟨normal⟩ messages and this needs addressing. After a node becomes a fake source there is a short delay before it starts to send messages $I_{TFS}(j)$ to help space out ⟨fake⟩ messages. This value was previously undefined for *DynamicSPR*, but now needs to be be deterministically specified. To simplify the implementation, it is set to be a constant for all nodes.

$$I_{TFS}(j) = \frac{P_{TFS}(j)}{4} \tag{7}$$

A TFS at node $j$ will send its $n^{\text{th}}$ ⟨fake⟩ message $\mathcal{F}_n^{TFS_j}$ during its duration as a fake source at the following times:

$$S_{TFS_j}(\mathcal{F}_n^{TFS_j}) = \tau_{TFS}(j) + I_{TFS}(j) + (n-1)P_{TFS}(j) \tag{8}$$

The set of times ⟨fake⟩ messages will be sent by TFS $j$ is:

$$S_{TFS_j}(\mathcal{F}^{TFS_j}) = \left\{ S_{TFS_j}(\mathcal{F}_n^{TFS_j}) \mid n \in [1 \,..\, \#_{TFS}(j)] \right\} \tag{9}$$

Thus we expect a node $k$ to receive ⟨Fake⟩ messages from the TFS at node $j$ at the following times:

$$R_k(\mathcal{F}^{TFS_j}) = \left\{ t + \alpha\Delta(k,j) \mid t \in S_{TFS_j}(\mathcal{F}^{TFS_j}) \right\} \tag{10}$$

The time between two fake nodes $j$ and $k$ sending their first ⟨fake⟩ message, where $j$ precedes $k$, is:

$$B(j,k) = S_j(\mathcal{F}_1^{TFS_k}) - S_i(\mathcal{F}_1^{TFS_j}) = D_{TFS}(j) + \alpha + I_{TFS}(k) - I_{TFS}(j) \tag{11}$$

As a TFS at $j$ moves to $k$ ($j$ becomes a normal node and $k$ becomes a TFS) its distance in hops with respect to another node $i$ can change. This change will be to increase the distance with respect to the source as the fake sources move away from the real source during the directed random walk, and there will be no change once a PFS has been allocated. However, the fake sources will get closer to other nodes in the network as they move away from the source. This means that the duty cycling needs to be able to handle messages arriving earlier and later than at the expected arrival time.

$$(\Delta(i, TFS_k) - \Delta(i, TFS_j)) \in \{ -1, 0, +1 \} \tag{12}$$

*4.5.1 TailFS Timings.* A TFS $j$ becomes a TailFS after its duration expires to ensure a reliable progression of fake sources.

$$\tau_{TailFS}(j) = \tau_{TFS}(j) + D_{TFS}(j) \tag{13}$$

A TailFS will send a ⟨fake⟩ message with same period and duration as a TFS after the same initial start delay. Essentially a TailFS is a TFS, but without a fixed duration. Once the duration period expires it will send another ⟨choose⟩ message to try to select the next TFS, but will remain a TailFS. Once a TailFS receives a message from a further TailFS or PFS, it will cease broadcasting ⟨fake⟩ messages and return to being a normal node.

$$D_{TailFS}(j) = D_{TFS}(j) \tag{14}$$

$$D_{TailFS}(j) = D_{TFS}(j) \tag{15}$$

$$D_{TailFS}(j) = D_{TFS}(j) \tag{16}$$

*4.5.2 PFS Timings.* The time at which the TFS $j$ hands off to a PFS $k$ is shown in Equation (17). The $\alpha$ component represents the time cost of sending a ⟨choose⟩ message from TFS $j$ to the node $k$ that becomes the PFS.

$$\tau_{PFS}(k) = \tau_{TFS}(j) + D_{TFS}(j) + \alpha \tag{17}$$

The PFS $k$ also has an initial start delay $I_{PFS}(k)$. To simplify the implementation this is set to the same constant used by both TFSs and TailFSs:

$$I_{PFS}(k) = I_{TailFS}(k) = I_{TFS}(k) = \frac{P_{TFS}(k)}{4} \tag{18}$$

A PFS $k$ will send its $n^{\text{th}}$ ⟨fake⟩ message $\mathcal{F}_n^{PFS_k}$ at the following times:

$$S_{PFS_k}(\mathcal{F}_n^{PFS_k}) = \tau_{PFS}(k) + I_{PFS}(k) + (n-1)P_{PFS}(k) \tag{19}$$

As the duration of a PFS is potentially unbounded, the set of all ⟨fake⟩ messages that the PFS $k$ could send is:

$$S_{PFS_k}(\mathcal{F}^{PFS_k}) = \left\{ S_{PFS_k}(\mathcal{F}_n^{PFS_k}) \mid n \in \mathbb{N}_1 \right\} \tag{20}$$

The time between each ⟨fake⟩ message from a PFS is its period $P_{TFS}(k)$. An issue is that the PFS period adjusts based on the percentage of ⟨fake⟩ messages the source has received ($\psi_{src}(\mathcal{F})$), a lower receive ratio at the source leads to a faster ⟨fake⟩ message generation rate at the PFS. The network is not expected to know when this change will occur or what the new period will be set to. Instead the change in the PFS period is required to occur slowly in order to allow the duty cycle to catch a ⟨fake⟩ message sent using the new period.

*4.5.3 Sleep Schedule.* Using these timings a sleep schedule can be developed for ⟨fake⟩ and ⟨choose⟩ messages. As with ⟨normal⟩ messages the nodes will wake up earlier and sleep later for both message types. This early wakeup and late sleep will also help account for the changing PFS period. The early wakeup and late sleep for ⟨choose⟩ messages will be represented as $EW(C)$ and $LS(C)$. The early wakeup and late sleep for ⟨fake⟩ messages will be represented as $EW(\mathcal{F})$ and $LS(\mathcal{F})$. There are four timing aspects that need to be kept track of:

(1) When the next ⟨fake⟩ message for the same TFS will be sent (considers the TFS period)
(2) When the next TFS will send its first ⟨fake⟩ message (considers the TFS duration)
(3) When a neighbouring TFS will send its ⟨choose⟩ message (considers the TFS duration)
(4) When a PFS will send its next ⟨fake⟩ message (considers the PFS period)

After receiving the $n^{\text{th}}$ ⟨fake⟩ message from $j$ at node $k$ the following knowledge about future message receives is known:

- The next ⟨fake⟩ message $n+1$ will be received at
$$R_k(\mathcal{F}_{n+1}^{TFS_j}) = R_k(\mathcal{F}_n^{TFS_j}) + P_{TFS}(j) \tag{21}$$
and every subsequent ⟨fake⟩ message from this TFS each $P_{TFS}(j)$ after.
- If the node did not receive the first ⟨fake⟩ message from a TFS for some reason. The time it should have been received can be calculated by:
$$R_k(\mathcal{F}_1^{TFS_j}) = R_k(\mathcal{F}_n^{TFS_j}) - (n-1)P_{TFS}(j) \tag{22}$$
- The ⟨choose⟩ from TFS $j$ at:
$$R_k(C) = R_k(\mathcal{F}_1^{TFS_j}) + D_{TFS}(j) - I_{TFS}(j) \tag{23}$$
- The first ⟨fake⟩ message from the next TFS $q$ is received at $k$ at:
$$R_k(\mathcal{F}_1^{TFS_q}) = R_k(\mathcal{F}_1^{TFS_j}) + D_{TFS}(j) - I_{TFS}(q) + I_{TFS}(j) + \alpha \tag{24}$$
As the initial delay is assumed to be constant (i.e., $I_{TFS}(q) = I_{TFS}(j)$), this simplifies to:
$$R_k(\mathcal{F}_1^{TFS_q}) = R_k(\mathcal{F}_1^{TFS_j}) + D_{TFS}(j) + \alpha \tag{25}$$

### 4.6 Timing Demonstration

To aid understanding the times at which events can occur, this section will explain a number of timing diagrams. These diagrams aim to show the standard cases that can occur, plus error cases and how the duty cycle handles these.

*4.6.1 TFS to TFS Hand-off.* In Figure 4 the timing of events for when a TFS $q$ hands off to a TFS $r$ is shown. The subsequent conversion of TFS $q$ into TailFS $q$ is omitted to maintain the simplicity of the diagram. The diagram shows ⟨fake⟩ messages being sent periodically after some initial start delay ($I_{TFS}(q)$). The ⟨fake⟩ messages are represented by solid arrows from the TFS $q$ or TFS $r$ to the normal node $j$. The normal node $j$ will receive messages from TFS $q$ with the same period between the messages ($\alpha\Delta(j,q)$). However, there is a gap when transitioning from TFS $q$ sending, to TFS $r$. This is due to the time cost ($\alpha$) of sending ⟨choose⟩ messages (represented by dashed arrows) plus the initial start delay ($I_{TFS}(r)$). Node $j$ calculates when to wake up for TFS $r$'s ⟨fake⟩ messages by considering the TFS duration.

*4.6.2 TFS to PFS Hand-off.* The diagram in Figure 5 is very similar to Figure 4 except that TFS $q$ hands off to a PFS $r$ via the ⟨choose⟩ message. Node $j$ can calculate when to expect PFS $r$'s first message using the duration of TFS $q$ because $I_{TFS}(q) = I_{PFS}(r)$.

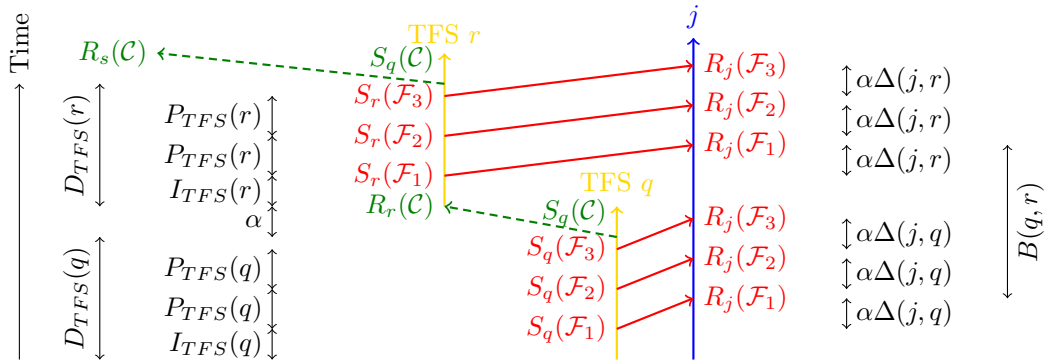



Fig. 4. Timing of events for two TFS $q$ and $r$ when $\#_{TFS}(j) = \#_{TFS}(k) = 3$. With ⟨fake⟩ messages represented by solid arrows and ⟨choose⟩ messages represented by dashed arrows.

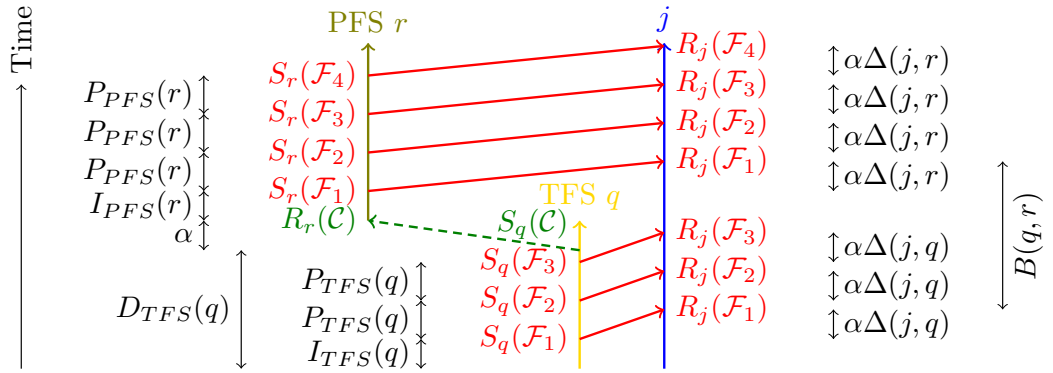



Fig. 5. Timing of events for a TFS $q$ and a PFS $r$ when $\#_{TFS}(q) = 3$. With ⟨fake⟩ messages represented by solid arrows and ⟨choose⟩ messages represented by dashed arrows.

*4.6.3 TFS to TFS Hand-off With Lost Messages.* The timing diagram in Figure 6 shows TFS $q$ becoming a TailFS and includes how it remains a TailFS until a ⟨fake⟩ message from a further node $r$ is received. In the first period of TFS $r$ the two messages sent are lost, perhaps due to collisions, high noise, or some other error. These lost messages are represented by solid lines that terminate with a circle. When TFS $r$ becomes a TailFS, it continues broadcasting with the same interval between ⟨fake⟩ messages after the initial start delay because $P_{TFS}(q) = P_{TailFS}(q)$. TailFS $q$ then receives this message and becomes a normal node since it has detected a further TailFS to take over its role. In this case the second round of ⟨fake⟩ messages are sent slightly earlier as there is no delay from sending a ⟨choose⟩ message. As this time is expected to be small it will be handled by the early wakeup and late sleep for ⟨fake⟩ messages. Note that in this scenario a second ⟨choose⟩ message was sent by TailFS $q$, since it was not aware that a further fake node had been created.

## 5 ALGORITHM

This section describes the duty cycle algorithm for *DynamicSPR-S*. An important aspect of these modifications are the early wakeup and late sleep for ⟨normal⟩, ⟨fake⟩ and ⟨choose⟩ messages and

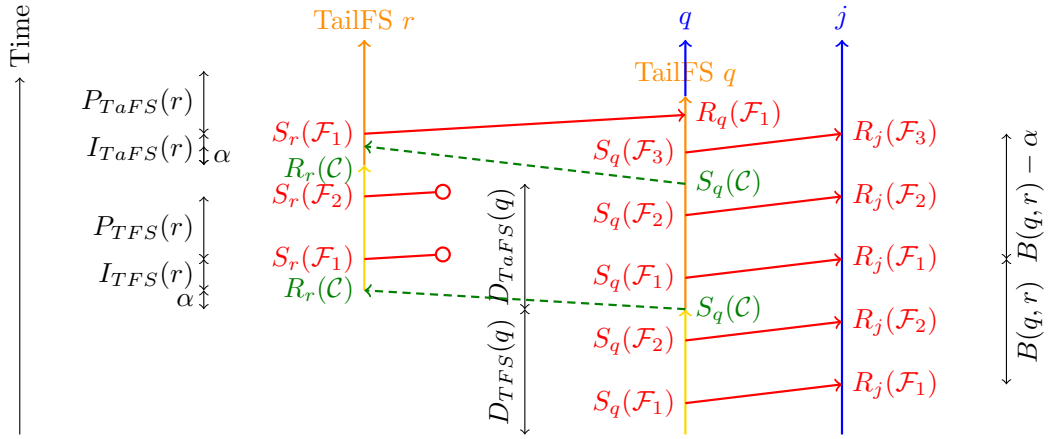
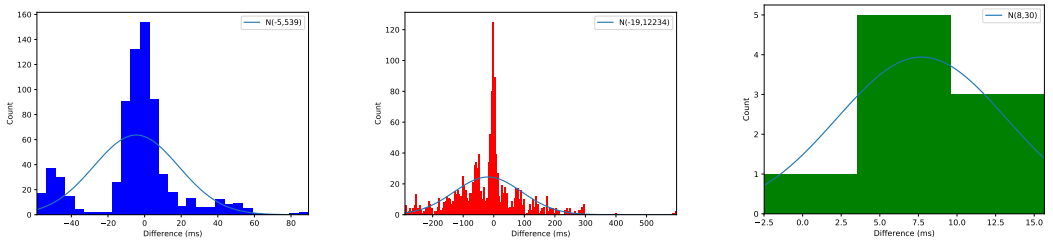



Fig. 6. Timing of events for a TFS $q$ that becomes a TailFS $q$ when $\#_{TFS}(q) = 2$ and $q$ fails to receive any ⟨fake⟩ messages from $r$ until its second duration. With ⟨fake⟩ messages represented by solid arrows and ⟨choose⟩ messages represented by dashed arrows.




(a) ⟨Normal⟩ (b) ⟨Fake⟩ (c) ⟨Choose⟩

Fig. 7. Histograms showing the receive time difference from the expected receive time on a $7 \times 7$ grid network when the early wakeup and late sleep are both large, with a fitted normal distribution curve.

what values should be used. Hence, before describing the algorithm implementation the values for the wakeup and sleep times are considered.

### 5.1 Early Wakeup and Late Sleep

Due to a variety of factors that there will be a difference between when messages are expected to be received and when they are actually received. For example, messages may not take the shortest path through the network due to collisions. This is a key reason why early wakeup ($EW(\mathcal{M})$) and late sleep ($LS(\mathcal{M})$) times are used. Different wakeup times will be specified for ⟨normal⟩ messages ($\mathcal{N}$), ⟨fake⟩ messages ($\mathcal{F}$), and ⟨choose⟩ messages ($C$).

An example of how the actual receive times differed from the expected receive time is shown in Figure 7 for a $7 \times 7$ grid where the distance between sink and source is 6 hops. Ideally the majority of messages should be received around a difference of 0 ms, however, there can be up to 120 ms between ⟨normal⟩ messages being delivered in this example. Results in real-world scenarios will be different, therefore, a way of specifying early wakeup and late sleep values is required.

**Algorithm 2** Duty Cycling Control

```
    ▷ Always start the radio when requested to
1:  event StartRadio() →
2:      signal StartRadioHardware()
    ▷ Only stop the radio if all components are not using it.
    ▷ The sink node never duty cycles. The source node ignores turn off rules for ⟨normal⟩ messages.
3:  event StopRadio() →
4:      if ¬IsSinkNode() ∧ (IsSourceNode() ∨ NormalCanTurnOff()) ∧ FakeCanTurnOff() ∧ ¬sending then
5:          signal StopRadioHardware()
```

**Algorithm 3** Send Duty Cycling

```
    sending ← 0
1:  event Send(msg) →
2:      signal StartRadio()
3:      SendOnRadio(msg)
4:      sending ← 1
5:  event SendDone() →
6:      sending ← 0
7:      signal StopRadio()
```

### 5.2 Implementation

The implementation of *DynamicSPR-S* requires a minor change to *DynamicSPR*. In *DynamicSPR*, when a ⟨fake⟩ message is sent, the protocol will retry to send it until it is successful. This is not performed when duty cycling is enabled, since it causes ⟨fake⟩ messages to go out of synchronisation with their expected times. If a node was to receive one of these delayed ⟨fake⟩ messages, it would expect to receive future ⟨fake⟩ messages at later times than they would actually be delivered.

The controls for the radio hardware are shown in Algorithm 2. When one component is finished using the radio, the component signals for the radio to be switched off. Only once no components are using the radio, is it switched off. When any single component asks for the radio to be switched on it is always switched on. When a node is sending a message it requests the radio be turned on and leaves it on until sending is complete as shown in Algorithm 3.

Duty cycling for ⟨normal⟩ messages in Algorithm 4 follows the Equations 4, 5, and 6. The usage of Equation (4) can be seen in StartOffTimerFromMessage(), Equation (5) can be seen in StartOffTimer(), and Equation (6) in StartOnTimer(). The duty cycling starts when a ⟨normal⟩ message is received. The time at which the radio received the message is recorded. It is important to use this time because the WSN needs the duty cycle to be anchored to when messages are actually received instead of when they are processed. If the current node time was used, then the duty cycling may fall out of synchronisation with respect to the source node that is sending messages.

When a ⟨normal⟩ message is received the OffTimer is started which times until the radio should turn off. When the OffTimer fires, the OnTimer is started which will turn the radio back on when the timer fires. As one ⟨normal⟩ message is expected to be received each time the radio is on, once that message has been received the radio can be turned off. When the radio is turned off in OffTimer the offEarly flag is reset. The radio can only turn off when OnTimer is running (the node is waiting to turn on) and OffTimer is not running (the node is not waiting to turn off), or when the radio can be turned off early.

The duty cycling for ⟨fake⟩ and ⟨choose⟩ messages is shown in Algorithm 5. The algorithm is event triggered from the first new ⟨fake⟩ message received. If the ⟨fake⟩ message arrives from a TFS or TailFS then the TempOff timer (to turn the radio off) is started, the DurationOn timer is started (if not running) which turns the radio on to receive a ⟨fake⟩ message from the next fake node, and the ChooseOn timer is started (if the node is adjacent to the fake source) to wake the node up to receive the ⟨choose⟩ message from the fake source. If the ⟨fake⟩ message arrives from a PFS then

**Algorithm 4** ⟨Normal⟩ Duty Cycling

```
    offEarly ← 0
    OnTimer, OffTimer ← ⊥, ⊥
 1: receive Normal⟨...⟩ →
 2:     now ← MessageReceiveTime()
 3:     if isNew() then
 4:         offEarly ← 1                          ▷ Turn off radio after receiving message
 5:         signal StopRadio()
 6:         StartOffTimerFromMessage(now)
 7: timeout (OnTimer) at now →
 8:     signal StartRadio()
 9:     StartOffTimer(now)
10: function StartOnTimer(now)
11:     if ¬IsRunning(OnTimer) then
12:         StartAt(OnTimer, now, P_src − EW(N) − LS(N))
13: function NormalCanTurnOff
14:     return offEarly ∨ (IsRunning(OnTimer) ∧ ¬IsRunning(OffTimer))
15: timeout (OffTimer) at now →
16:     offEarly ← 0
17:     signal StopRadio()
18:     StartOnTimer(now)
19: function StartOffTimer(now)
20:     if ¬IsRunning(OffTimer) then
21:         StartAt(OffTimer, now, EW(N) + LS(N))
22: function StartOffTimerFromMessage(now)
23:     if ¬IsRunning(OffTimer) then
24:         StartAt(OffTimer, now, LS(N))
```

the PermOff timer is started. The offEarly flag is set to allow the radio to turn off immediately and the radio is signalled to stop because no other ⟨fake⟩ message from a PFS is expected. Starting the PermOff timer leads to the PermOn timer being fired after the PFS period.

When receiving a ⟨fake⟩ message, if it has not come from a TFS or TailFS, TempNoReceive is incremented, if the message has come from a TFS or TailFS then it is reset. If three rounds of ⟨fake⟩ messages are missed from TFSs or TailFSs, then both the TempOff and TempOn timers are stopped. The reason for this is that once a PFS has been allocated, TFSs and TailFSs should revert to being normal nodes. To save energy the awake periods reserved for ⟨fake⟩ messages from TFSs and TailFSs are no longer needed, so these timers (which perform the wakeups) can be stopped.

The radio can be turned off for this component when all of the following are true: (i) it is not in the receive window for a ⟨fake⟩ message from a TFS or TailFS, or receiving a ⟨fake⟩ message from these node types is disabled, (ii) it is not the receive window for a ⟨fake⟩ message from a PFS or the radio can be turned off early as this ⟨fake⟩ has been received, and (iii) it is not in the receive window for a ⟨choose⟩ message.

## 6 EXPERIMENTAL SETUP

In this section we describe the simulation environment and protocol configurations that were used to generate the results presented in Section 7. A combination of simulations and experiments on real-world hardware are used to demonstrate the efficacy of *DynamicSPR-S*. Simulations are primarily used to explore the impact of different network sizes on the protocol's performance and test with many different combinations of parameters to the developed duty cycle and TinyOS' LPL. A deployment on the FlockLab testbed is used to test performance in a real-world radio environment, to obtain current draw measurements and on a non-grid network topology.

### 6.1 Simulation Environment and Network Configuration

The protocol[1] was implemented using TinyOS [28] and simulated using COOJA [38]. COOJA is a cycle accurate WSN simulator that accurately models hardware in order to execute compiled binaries for hardware platforms. It is also capable of modelling the communications between sensor

[1]Source code for the protocol and analysis scripts available at: https://github.com/MBradbury/slp

**Algorithm 5** ⟨Fake⟩ and ⟨Choose⟩ Duty Cycling

```
    offEarly, tempDisabled, tempNoReceive ← 0, 0, 0
    ChooseOn, ChooseOff, DurationOn, DurationOff ← ⊥, ⊥, ⊥, ⊥      ▷ Timers are initialised to not fire at any time
    TempOn, TempOff, PermOn, PermOff ← ⊥, ⊥, ⊥, ⊥
    ▷ ult_fake_cnt is 0 for the first ⟨fake⟩ message
 1: function RecvFromTempOrTail(now, ult_fake_cnt)
 2:     if isNew() then
 3:         StartTempOffTimerFromMessage(now)
            ▷ How long ago the first ⟨fake⟩ from this TFS/TailFS was sent
 4:         d ← P_TFS × ult_fake_cnt
 5:         if ¬IsRunning(DurationOn) then
 6:             StartAt(DurationOn, now, D_TFS − d − EW(F))
 7:         if isAdjacent() ∧ ¬IsRunning(ChooseOn) then
 8:             StartAt(ChooseOn, now, D_TFS − I_TFS − d − EW(C))
 9: function RecvFromPerm(now)
10:     if isNew() then
11:         StartPermOffTimerFromMessage(now)
12:         offEarly ← 1                ▷ Turn off radio after receiving msg
13:         signal StopRadio()
14: receive Fake⟨src_type, ult_fake_cnt, ...⟩ →
15:     now ← MessageReceiveTime()
16:     if src_type ∈ { TempFake, TailFake } then
17:         RecvFromTempOrTail(now, ult_fake_cnt)
18:     else if src_type ∈ { PermFake } then
19:         RecvFromPerm(now)
20:     if ¬tempDisabled then
21:         if IsRunning(TempOff) ∧ ¬IsRunning(TempOn) then
22:             tempNoReceive ← 0
23:         else
24:             tempNoReceive ← tempNoReceive + 1
25:             if tempNoReceive ≥ 3 then
26:                 tempDisabled ← 1
27:                 Stop(TempOff)
28:                 Stop(TempOn)
```

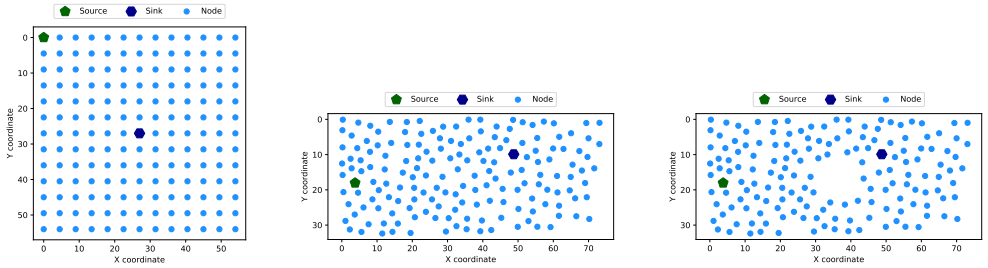
**Algorithm 6** ⟨Fake⟩ and ⟨Choose⟩ Duty Cycling: On and Off Timers

```
29: timeout (DurationOn) at now →
30:     signal StartRadio()
31:     StartAt(DurationOff, now, EW(F) + LS(F))
32: timeout (TempOn) at now →
33:     signal StartRadio()
34:     StartTempOffTimer(now)
35: timeout (PermOn) at now →
36:     signal StartRadio()
37:     StartPermOffTimer(now)
38: timeout (ChooseOn) at now →
39:     signal StartRadio()
40:     StartAt(ChooseOn, now, EW(C) + LS(C))
41: timeout (DurationOff) at now →
42:     signal StopRadio()
43:     Stop(TempOn)
44:     StartTempOn(now)
45: timeout (TempOff) at now →
46:     signal StopRadio()
47:     StartTempOn(now)
48: timeout (PermOff) at now →
49:     offEarly ← 0
50:     signal StopRadio()
51:     StartPermOn(now)
52: timeout (ChooseOff) →
53:     signal StopRadio()
```

**Algorithm 7** ⟨Fake⟩ and ⟨Choose⟩ Duty Cycling

```
54: function StartTempOn(now)
55:     if ¬IsRunning(TempOn) then
56:         StartAt(TempOn, now, P_TFS − EW(F) − LS(F))
57: function StartTempOffTimer(now)
58:     if ¬IsRunning(TempOff) then
59:         StartAt(TempOff, now, EW(F) + LS(F))
60: function StartTempOffTimerFromMessage(now)
61:     if ¬IsRunning(TempOff) then
62:         StartAt(TempOff, now, LS(F))
63: function StartPermOffTimer(now)
64:     if ¬IsRunning(PermOff) then
65:         StartAt(PermOff, now, EW(F) + LS(F))
66: function StartPermOffTimerFromMessage(now)
67:     if ¬IsRunning(PermOff) then
68:         StartAt(PermOff, now, LS(F))
69: function FakeCanTurnOff
70:     return ((IsRunning(TempOn) ∧ ¬IsRunning(TempOff)) ∨ tempDisabled) ∧ (offEarly ∨ ¬IsRunning(PermOff)) ∧ ¬IsRunning(ChooseOff)
```

nodes. A square grid network layout of size $n \times n$ was used in the majority of experiments, where $n \in \{7, 9, 11, 13\}$, i.e., networks with 49, 81, 121 and 169 nodes respectively. The node neighbourhoods were modelled using the unit disk graph model (UDGM) with a communication range of 4.75 m, and nodes were located 4.5 m away from their north, south, west and east neighbours. A single




(a) 13 × 13 SourceCorner (b) PoissonDiskConnected (c) PoissonDiskWithHole

Fig. 8. The three simulated network configurations

source node generated messages and a single sink node collected messages. The source and sink nodes were distinct and assigned positions in the SourceCorner configuration from [24] where the source is located in the top left corner of the network and the sink in the centre. The rate at which messages from the real source were generated was varied and is given by $P_{src} \in \{0.5, 1, 2, 4, 8\}$ seconds. At least 200 repeats were performed for each combination of parameters.

For a small number of simulations a network with a random layout was generated using Bridson's algorithm [8] for Poisson Disk Sampling, with and without a hole between the sink and source. PoissonDiskConnected has 169 nodes and PoissonDiskWithHole was derived from PoissonDiskConnected by removing nodes that were 6.75 m from the centre point of the network. Simulations for these configurations instead use the LogisticLoss Radio Model [16] which uses a log-distance path loss model. Nodes had a communication range of 4.75 m, otherwise the default parameters to LogisticLoss were used. These configurations are used to demonstrate the applicability of the presented techniques to unstructured networks with a different communication model. The SourceCorner grids are necessary to evaluate the impact of varying network sizes on the performance of the proposed techniques, as their regular structure allows expansion to different network sizes.

### 6.2 Comparison with Phantom Routing

A comparison with Phantom Routing is performed, where it is tested with TinyOS's LPL. Phantom Routing has two main parameters, the landmark node which messages are routed towards or away from and the maximum phantom path length. In these experiments the landmark node is set to be the top right corner of the network and the maximum phantom path length is set to the sink-source distance, so size 7, 9 and 11 networks will respectively have a maximum phantom path length of 6, 8 and 10 hops. The same TinyOS LPL parameters used for *DynamicSPR* are used for Phantom Routing. At least 250 repeats were performed for simulations of Phantom Routing.

As the original description of Phantom Routing did not consider several practical implementation aspects, changes have been made to Phantom Routing to improve its performance[2]. The first major change is that retransmissions are performed along the directed random walk to reduce packet loss during duty cycling. The second is to terminate the random walk early, when no further valid nodes exist to continue the directed random walk, by beginning the flood of messages from the terminating node. The third is that when choosing which direction that walk should go (towards or away from the landmark node), experiments found that bias in the random number generated by MLCG (TinyOS' PRNG) caused one direction to be used more than the other leading to higher capture ratios. This implementation samples from a higher bit in order to mitigate this bias.

[2]Implementation available at: https://github.com/MBradbury/slp/tree/master/algorithm/phantom

| Network Size | Source Period (seconds/message) 8.0 | 4.0 | 2.0 | 1.0 | 0.5 |
|---|---|---|---|---|---|
| $7 \times 7$ | 78.13 | 39.06 | 19.53 | 9.77 | 5.03 |
| $9 \times 9$ | 110.00 | 54.85 | 27.39 | 13.76 | 7.07 |
| $11 \times 11$ | 140.63 | 70.31 | 35.16 | 17.58 | 9.01 |
| $13 \times 13$ | 172.94 | 86.55 | 43.25 | 22.07 | 11.08 |

Table 3. Safety Period (in seconds) for the Source-Corner configuration in the COOJA simulator using the UDGM radio model.

| Source Period | 1.0 | 2.0 | 8.0 |
|---|---|---|---|
| Safety Period | 8.82 | 17.02 | 76.18 |

Table 4. Safety Period (in seconds) for the FlockLab testbed.

| Configuration | Source Period (seconds/message) 8.0 | 4.0 | 2.0 | 1.0 | 0.5 |
|---|---|---|---|---|---|
| PoissonDiskConnected | 444.43 | 220.20 | 109.66 | 56.26 | 27.67 |
| PoissonDiskWithHole | 980.46 | 463.05 | 239.65 | 118.29 | 58.26 |

Table 5. Safety Period (in seconds) for two random network configurations in the COOJA simulator using the LogisticLoss radio model.

| | $t_s$ (ms) | $t_{tx}$ (ms) | $t_d$ (ms) | $cca$ |
|---|---|---|---|---|
| 1 | 50 | 50 | 10 | 1150 |
| 2 | 50 | 50 | 100 | 400 |
| 3 | 75 | 75 | 10 | 1150 |
| 4 | 75 | 75 | 10 | 2300 |
| 5 | 75 | 75 | 100 | 400 |

Table 6. Parameters for TinyOS LPL

| | $EW(\mathcal{N})$ | $LS(\mathcal{N})$ | $EW(\mathcal{F})$ | $LS(\mathcal{F})$ | $EW(C)$ | $LS(C)$ |
|---|---|---|---|---|---|---|
| 1 | 200 | 200 | 250 | 250 | 75 | 75 |
| 2 | 80 | 80 | 120 | 130 | 5 | 50 |
| 3 | 40 | 40 | 120 | 130 | 5 | 50 |
| 4 | 35 | 35 | 100 | 100 | 5 | 50 |
| 5 | 35 | 35 | 60 | 60 | 5 | 50 |

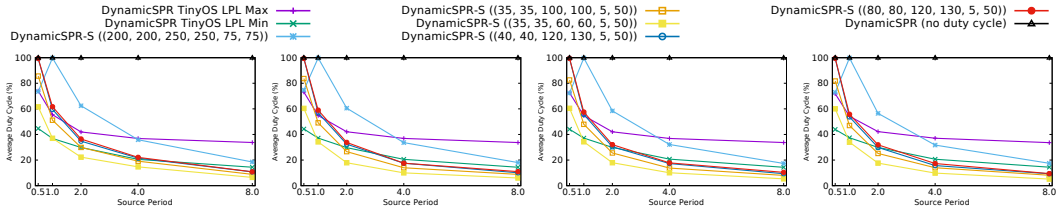
Table 7. Wakeup intervals for *DynamicSPR* in milliseconds

### 6.3 Safety Period

As flooding has been shown to provide no SLP (as identified in the seminal work [25]) we use the average time it takes the attacker to capture the source as a baseline and double it to calculate the safety period (shown in Tables 3 and 5 for COOJA, and Table 4 for FlockLab) for the protocol in this paper. Flooding provides no SLP as it allows an attacker to trace back to the location of the source hop-by-hop each time the source broadcasts.

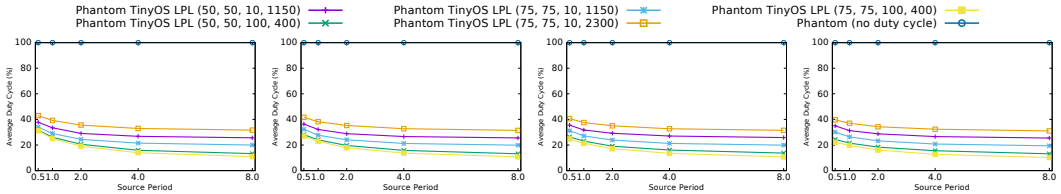
### 6.4 Simulation Experiments

An experiment constituted a single execution of the simulator using a specified protocol configuration, network size, and safety period. An experiment finished when the source node had been captured by an attacker or the safety period elapsed. An attacker was implemented based on the log output from COOJA. It maintains internal state of its location using node identifiers. When a node receives a message, if the attacker is at the same location it will move based on the attacker model specified in Section 3.3. The *DynamicSPR* algorithm being tested has one parameter which specifies if one or two messages should be sent per source period, these are called `Fixed1` and `Fixed2`. The duty cycle for *DynamicSPR* has 6 parameters and the 5 combinations of these parameters investigated are shown in Table 7. Experiments were also run using the TinyOS Low Power Listening duty cycle implementation for comparison, with the parameters shown in Table 6. The default values for *msbd* and $t_{ack}$ are used.




(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 9. *DynamicSPR* Variants: The Average Duty Cycle when varying wakeup intervals and network sizes.




(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 10. *Phantom Routing* Variants: The Average Duty Cycle when varying TinyOS LPL parameters and networks sizes.

## 7 RESULTS

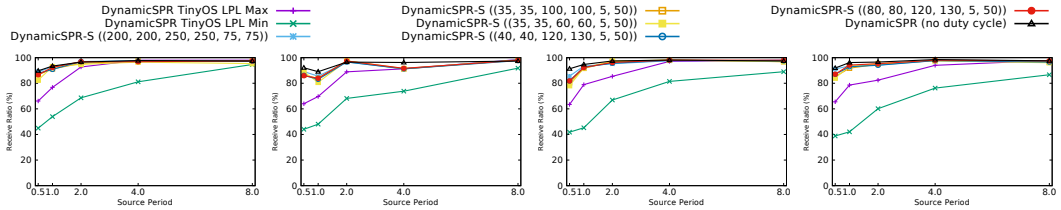
This section presents the results for *DynamicSPR*, the *DynamicSPR-S* duty cycle algorithm, *DynamicSPR* using TinyOS LPL for comparison and *Phantom Routing* with and without using TinyOS LPL. The COOJA simulator was used for all simulated experiments as it is capable of simulating MAC layers. Only the `Fixed1` and `Fixed2` approaches of *DynamicSPR* was simulated and not the `Rnd` approach. This is because *DynamicSPR-S* only supports deterministic approaches. Only the results for the `Fixed2` approach are presented for brevity.

Results are shown for wakeup intervals in the form of this 6-tuple: $(EW(\mathcal{N}), LS(\mathcal{N}), EW(\mathcal{F}), LS(\mathcal{F}), EW(C), LS(C))$, which includes the early wakeup and late sleep values for ⟨normal⟩ messages ($\mathcal{N}$), ⟨fake⟩ messages ($\mathcal{F}$), and ⟨choose⟩ messages ($C$). Five different sets of wakeup intervals are simulated (shown in Table 7), to test the effects of different duty cycles on the selected metrics. The graphs include results for *DynamicSPR* without duty cycling enabled, to allow a comparison which indicates performance differences due to duty cycling. The experiments allowed various metrics of the performance of *DynamicSPR* to be collected. The following metrics will be analysed in Sections 7.1 to 7.4 for *DynamicSPR-S*, compared with TinyOS LPL in Section 7.5, compared with *Phantom Routing* in Section 7.6, and evaluated for experiments using the LogisicLoss radio model in Section 7.7 and experiments run on FlockLab in Section 7.8:

(1) **Duty Cycle** — The average percentage of time that the radio was on.
(2) **Received Ratio** — The percentage of messages that were sent by the source and received by the sink.
(3) **Average Number of Messages Sent** — The average number of messages sent across all nodes per second.
(4) **Capture Ratio** — The percentage of runs in which the attacker reaches the location of the source, i.e., captures the source, within the safety period.

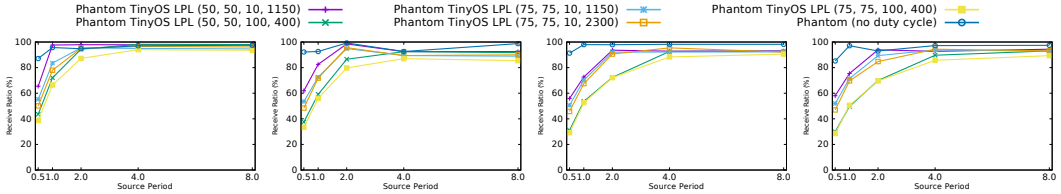
### 7.1 Duty Cycle

The graphs in Figure 9 show that the duty cycle technique for *DynamicSPR* reduces the amount of time that the radio is on. Smaller wakeup intervals lead to a lower duty cycle. Decreasing the ⟨fake⟩




(a) 7 × 7 (b) 9 × 9 (c) 11 × 11 (d) 13 × 13

Fig. 11. *DynamicSPR* Variants: The Received Ratio when varying wakeup intervals and network sizes.




(a) 7 × 7 (b) 9 × 9 (c) 11 × 11 (d) 13 × 13

Fig. 12. *Phantom Routing* Variants: The Received Ratio when varying TinyOS LPL parameters and networks sizes.

message wakeup intervals lead to a larger decrease as more ⟨fake⟩ messages are sent than ⟨normal⟩ messages. The *DynamicSPR* and *Phantom Routing* without duty cycling result shows a duty cycle of 100% because the radio is left on permanently. A larger source period led to smaller duty cycles because fixed sized wakeups are used. When fewer messages are sent it means that there is a reduction in the number of radio wakeups that need to be performed. Since fewer absolute wakeups are performed over the same period of time, the ratio of awake time to asleep time decreases.

### 7.2 Received Ratio

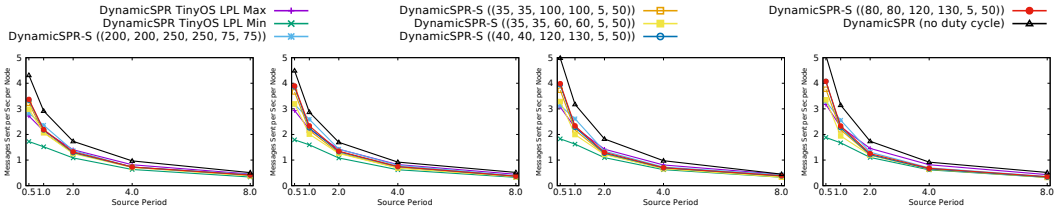
The received ratio shown in Figure 11 is above 80%. The received ratio increases for larger source periods, as fewer messages are sent over the same period of time thus reducing the chance of message losses due to collisions. Larger networks have higher receive ratios. Due to the increase in the number of paths available for messages to travel, a collision has less effect in larger networks.

The results for *DynamicSPR* without duty cycling have a higher received ratio compared to when duty cycling is enabled. This is because there is no chance for a ⟨normal⟩ message to be lost because a node's radio is off. Smaller wakeup intervals have a lower received ratio for the opposite reason, since there is a greater chance for messages to be missed because the radio is off for longer.

### 7.3 Messages Sent

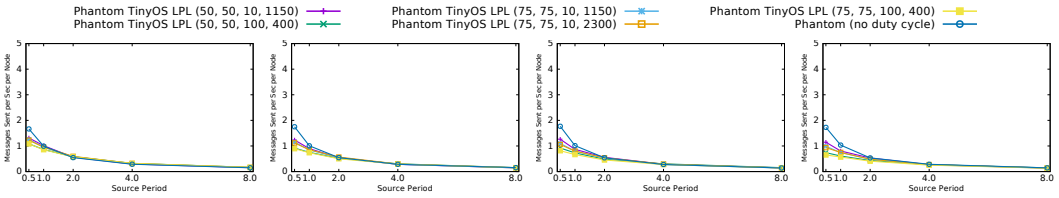
In Figure 13 it can be seen that fewer messages are sent when duty cycling is enabled compared to when it is disabled. This is because when duty cycling is enabled there is the possibility that a message might be sent outside the range the target node is awake. The reason that there appears to be a large difference between when duty cycle is enabled and when it is disabled, is because if a message is lost early in its path then it will not be received to be forwarded later.

Lower wakeup intervals lead to a lower number of messages sent. As there is less time when a node is awake, the probability of it receiving a message is lower. However, there only tends to be a small difference in the quantity of messages sent between the different wakeup interval parameters.



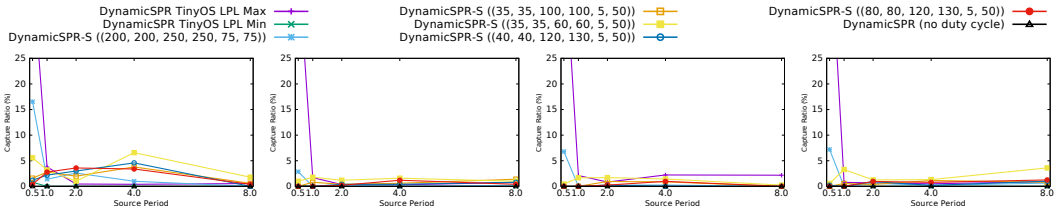

(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 13. *DynamicSPR* Variants: The total messages sent per node per second when varying wakeup intervals and network sizes.



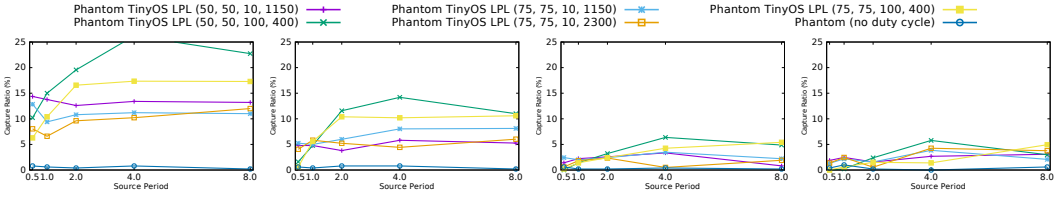

(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 14. *Phantom Routing* Variants: The total messages sent per node per second when varying TinyOS LPL parameters and networks sizes.




(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 15. *DynamicSPR* Variants: The Capture Ratio when varying wakeup intervals and network sizes.




(a) $7 \times 7$ (b) $9 \times 9$ (c) $11 \times 11$ (d) $13 \times 13$

Fig. 16. *Phantom Routing* Variants: The Capture Ratio when varying TinyOS LPL parameters and networks sizes.

### 7.4 Captured

The final metric that will be investigated is the capture ratio shown in Figure 15. When enabling duty cycling a higher capture ratio is observed, with different increases for different parametrisations. It was expected that smaller networks would experience a larger capture ratio than larger networks, as this pattern was previously reported. This result was indeed observed for the $7 \times 7$ scenario, as it has higher capture ratios than the larger $9 \times 9$ and $11 \times 11$ networks. The reason for this is that

there is a greater distance between the sink and source, which provides more time for *DynamicSPR* to recover if the attacker makes moves towards the source.

In most cases, when the duty cycle had larger wakeup intervals, the results show that the capture ratio is lower. This is because fewer messages are lost due to the radio being off. This means that there is a trade-off to be made between power consumption and SLP. To provide better reliability when forwarding a message and thus a lower capture ratio, a node will need to wakeup for a longer period. This is due to the difference in the time it takes for messages to travel through the network.

### 7.5 Comparison With TinyOS LPL

The duty cycle developed using knowledge of how *DynamicSPR* operates performs better than TinyOS LPL. The delivery ratio is worse for high source periods, but improves as the rate of sending messages slows. The capture ratio remains similar to *DynamicSPR-S*, but the duty cycle provided by TinyOS LPL is outperformed by *DynamicSPR-S* for source periods of 1.0 s and higher. The reason for this is that *DynamicSPR-S* takes advantage of how *DynamicSPR* operates and then uses fixed length wakeups to send and receive messages. This is in comparison to TinyOS LPL which periodically wakes up even if there is nothing to be done. As the source period increases (and the rate of sending messages slows) *DynamicSPR-S* will have a lower duty cycle due to the fixed length wakeups.

### 7.6 Comparison with *Phantom Routing*

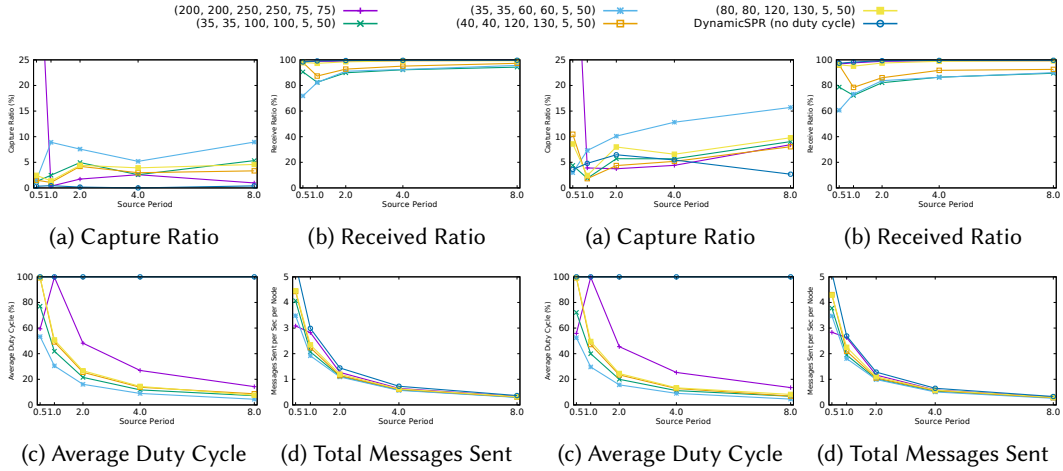
Comparing the results for *Phantom Routing* using TinyOS LPL to *DynamicSPR* using TinyOS LPL and *DynamicSPR-S* show that *Phantom Routing* is not well suited to having an arbitrary duty cycle applied to it. In Figure 10 the duty cycle obtained is less than or equal to 40% with some parameterisations reaching 10%. The minimum and maximum duty cycles were lower for *Phantom Routing* than for *DynamicSPR* due to the smaller number of messages sent by *Phantom Routing* as shown in Figure 14. One of the main advantages of *Phantom Routing* over fake source based techniques is that fewer messages need to be sent compared to *DynamicSPR*. Our results show that *Phantom Routing* requires between 2 and 5 times fewer messages sent per node per second.

While *Phantom Routing* outperforms *DynamicSPR* and its duty cycled variants in terms of the duty cycle and number of messages sent, it delivers worse results for the received ratio and capture ratio. For *Phantom Routing* in Figure 12 the received ratio worsens as the network size (and thus the sink-source distance) increases. The received ratio is highest when no duty cycling is used and is lower for TinyOS LPL parameters that lead to a lower duty cycle, indicating that the time spent sleeping is leading to lost messages. Typically, a lower receive ratio also leads to a lower capture ratio as fewer messages reach a location near the sink and are eavesdropped by the attacker. However, *Phantom Routing*'s ability to prevent the attacker from capturing the source when using TinyOS LPL tends to be worse than both *DynamicSPR* using TinyOS LPL and *DynamicSPR-S*.

This shows that TinyOS LPL, an SLP-unaware duty cycle, leads to performance degradation in terms of SLP when applied to *Phantom Routing*. We conjecture that other WSN duty cycle protocols (such as ContikiMAC [14]) will lead to similar results as they alter temporal and spatial aspects of packet routing which SLP-aware routing protocols are attempting to control in a specific manner.

### 7.7 Simulations with Random Layout and LogisticLoss Radio Model

Results for simulations using COOJA and the LogisticLoss radio model for the PoissonDiskConnected and PoissonDiskWithHole configurations are shown in Figures 17 and 18 respectively. These results are included to demonstrate the performance of *DynamicSPR-S* on unstructured networks with a more realistic radio model. These results show similar patterns to the results obtained on a grid network with a UDGM radio model in terms of the patterns observed in the received ratio, average duty cycle and number of messages sent metrics. The main difference is in terms of the




(a) Capture Ratio (b) Received Ratio (c) Average Duty Cycle (d) Total Messages Sent

Fig. 17. Results comparing *DynamicSPR-S* and *DynamicSPR* on PoissonDiskConnected using the LogisticLoss radio model

(a) Capture Ratio (b) Received Ratio (c) Average Duty Cycle (d) Total Messages Sent

Fig. 18. Results comparing *DynamicSPR-S* and *DynamicSPR* on PoissonDiskWithHole using the LogisticLoss radio model

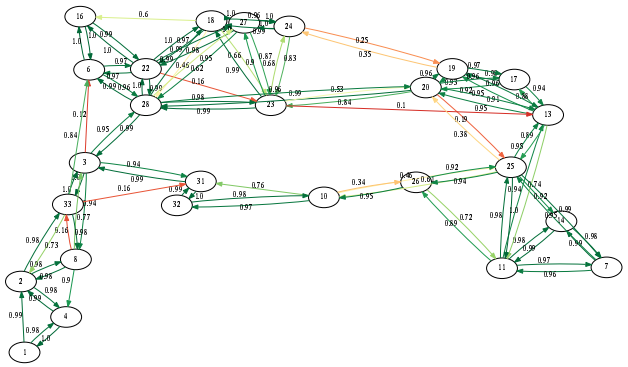
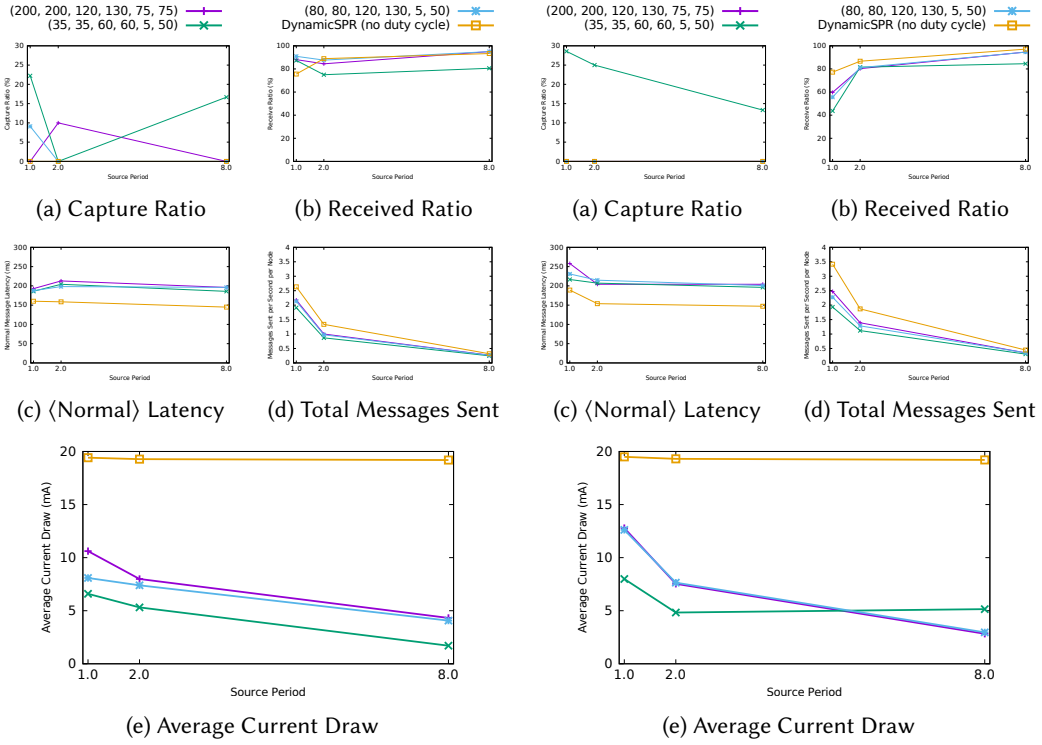


Fig. 19. FlockLab topology labelled with the probability of receiving a message along a directional link [7].

capture ratio which when compared to results in a observed in a grid network is up to 10 percentage points higher for PoissonDiskConnected, 15 percentage points higher for PoissonDiskWithHole, and 10 percentage points higher for PoissonDiskWithHole when excluding the *DynamicSPR-S* parameters with the smallest wake up window. The worse performance with PoissonDiskWithHole is partly caused by the introduction of a hole in the network as demonstrated by *DynamicSPR* (with no duty cycle) having a higher capture ratio in Figure 18a compared to Figure 17a. However, the use of the LogsticLoss radio model or unstructured network topology has also had an impact on the performance of *DynamicSPR*. Whilst there is a worsening of the capture ratio and receive ratio, the routing protocol still provides SLP, future work is needed to address the performance loss due to complex network topologies.

### 7.8 Real World Experimental Results (via FlockLab)

*DynamicSPR* and *DynamicSPR-S* were also deployed on the FlockLab [29] testbed with the source at node 1 and the sink at node 23 in the topology shown in Figure 19. The testbed is equipped with 27 TelosB nodes. The results for the duty cycle, receive ratio, and capture ratio showed similar patterns to those obtained when simulating on COOJA. The testbed has the ability to record the




(a) Capture Ratio (b) Received Ratio (c) ⟨Normal⟩ Latency (d) Total Messages Sent (e) Average Current Draw

Fig. 20. Results for *DynamicSPR* and *DynamicSPR-S* Fixed1 on the FlockLab testbed

(a) Capture Ratio (b) Received Ratio (c) ⟨Normal⟩ Latency (d) Total Messages Sent (e) Average Current Draw

Fig. 21. Results for *DynamicSPR* and *DynamicSPR-S* Fixed2 on the FlockLab testbed

current draw of the nodes, of which the average is presented in Figures 20e and 21e. These results show that a 37% reduction in current draw[3] can be achieved with duty cycling *DynamicSPR* when 1 message is sent by the source every second, and an 80% reduction is achievable when a message is sent every 8 seconds. This means that the duty cycle approach presented in this paper is capable of producing large energy usage saving, making *DynamicSPR-S* suitable for deployment.

## 8 DISCUSSION

One of the main changes that was made to *DynamicSPR* was to set TailFSs to use parameters from TFSs. Previously [6] TailFSs used the parameters from PFSs because TailFSs have a potentially unbounded duration. However, this parametrisation did not work for this duty cycle because the PFS period changes depending on the percentage of ⟨fake⟩ messages received at the source. These TailFSs led the duty cycle to go out of sync. By using TFS timings this issue was addressed.

### 8.1 Handling Clock Drift

Over time a sensor node's clock will drift. For a 32.768 kHz oscillator with a clock stability of ±20ppm, time could drift 52.7 ms a day. This drift is slow enough to not be an issue over a small number of messages as each node has a large wakeup period to allow it to handle difference in the

[3]Power consumption could be calculated by multiplying the current draw by the expected voltage of 3.3 V supplied to the sensor nodes. However, as FlockLab does not log the voltage along with the current draw we cannot be sure of the voltage and therefore power consumption at a specific time [7].

times at which a message may be delivered at a node. However, over time this drift could cause issues for which there are two solutions. The first is to perform some form of clock synchronisation (such as FTSP [33]), however would add an extra energy cost when running *DynamicSPR-S*. An alternate approach would be to periodically reset the time from which subsequent wakeups are computed in relation to. In this implementation all wakeups are computed relative to the first ⟨normal⟩ and ⟨fake⟩ received. Instead the wakeup calculation could be reset and calculated from a different message at certain times. Another approach would adjust the early wakeup and late sleep each time a message is received. The aim would be to keep the next message's arrival time in the middle of the wakeup period. This would be able to adjust for latency changes and clock drift.

As *DynamicSPR-S* is triggered by the asset detection, clock synchronisation only needs to be considered during the time *DynamicSPR-S* is active. Outside of this time, other duty cycles and clock synchronisation techniques should be used. When choosing a maximum time for the activation of *DynamicSPR-S* for an asset, the potential clock drift during that time should be used to inform values for early wakeups and late sleeps. Future work could evaluate the performance of increasing the awake window the longer *DynamicSPR-S* runs. This approach to handle clock drift is *best-effort* and so if stronger guarantees are needed clock synchronisation algorithms should be run during the execution of *DynamicSPR-S*.

### 8.2 Applicability to Other Techniques

In this paper, we viewed the superposition of a duty cycle atop an SLP protocol as a transformation process. We proposed two rules, based on message transmissions, for the transformation. We conjecture that the transformation process can be applied to other *spatial* SLP protocols [23] modelled as schedules, as shown in Figures 20 and 21, where two different variants of *DynamicSPR* were considered. As such, once the non-duty cycled SLP protocol has been modelled as a schedule, then similar transformation rules can be applied, with the difference being in the size of the windows. The sizes will be dependent, among others, on environmental factors such as line of sight communication, noise levels, etc. However, such transformation process cannot be used for temporal SLP protocols (e.g. [5]) as the protocols cannot be adequately modelled as a periodic schedule since nodes intentionally delay message transmissions. Nor can this approach be applied to techniques that either react nondeterministically to events or undertake proactive protection. As temporal SLP techniques induce delays intentionally, the use of existing duty cycle techniques (e.g., TSCH [15]) may be applicable, but further work is needed to investigate their impact.

### 8.3 Multiple Sources

The technique proposed selects a set of nodes to act as fake sources. Specifically, it is the case that a node selected as fake source did not previously act as a message relay. In the case of multiple sources, this issue is challenging as a node may be a relay for one source and a fake source for another. Thus, this interference needs to be factored in when choosing fake sources. Removing nodes which act as a relay reduces the size of the set of potential fake sources, making it nearly impossible when the number of sources is large because there is insufficient spatial redundancy [23]. In these cases, using a temporal SLP-aware routing protocol, such as [5], is advisable.

### 8.4 Minimising Power State Transitions

In this work we have developed a duty cycle that focused on reducing the time nodes spend awake while continuing to ensure a high level of SLP. However, there is also a cost to these transitions between power states in terms of both energy and time [9]. Future improvements to duty cycles for spatial SLP-routing protocols should consider these costs and factor the wakeup time of the specific radio used on a sensor into the calculations of when to wakeup and sleep. Future work

could also consider the cost related to the number of sleep-state transitions and also design a duty cycle which aims to avoid transitioning between sleep states if not beneficial.

## 9 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a novel formalisation of duty cycling as a transformation process. We then presented scheduling model for *DynamicSPR* and transformed the schedule via a timing analysis to obtain a duty cycled *DynamicSPR* (*DynamicSPR-S*). Different early wakeup and late sleep intervals have been investigated for different message types. Simulations and real-world experiments on the FlockLab testbed show that significant energy savings are achieved at the expense of a relatively very small decrease in SLP levels. For future work we plan to prove the correctness of the transformation and this duty cycle protocol. We also intend to investigate dynamically determining the wakeup parameters in response to a changing network environment.

## ACKNOWLEDGMENTS

This research was supported by the Engineering and Physical Sciences Research Council (EPSRC) through [DTG Grant EP/M506679/1] and the PETRAS: UK Research Hub for Cyber Security of the Internet of Things project [EPSRC Grant EP/N02334X/1].

## REFERENCES

[1] Alina-Mihaela Badescu and Lucian Cotofana. 2015. A wireless sensor network to monitor and protect tigers in the wild. *Ecological Indicators* 57 (2015), 447–451. https://doi.org/10.1016/j.ecolind.2015.05.022

[2] Zinaida Benenson, Peter M. Cholewinski, and Felix C. Freiling. 2008. *Wireless Sensors Networks Security*. Vol. 1. IOS Press, Amsterdam, The Netherlands, Chapter Vulnerabilities and Attacks in Wireless Sensor Networks, 22–43.

[3] Asset Berdibek and Sain Saginbekov. 2019. A Routing Protocol for Source Location Privacy in Wireless Sensor Networks with Multiple Sources. In *Proceedings of the 15th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet'19)*. ACM, New York, NY, USA, 93–99. https://doi.org/10.1145/3345837.3355951

[4] Johannes Blobel and Falko Dressler. 2017. Sender-Triggered Selective Wake-Up Receiver for Low-Power Sensor Networks. In *36th Conference on Computer Communications Workshops*. IEEE, Atlanta, GA, 984–985.

[5] Matthew Bradbury and Arshad Jhumka. 2017. A Near-Optimal Source Location Privacy Scheme for Wireless Sensor Networks. In *16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, Sydney, Australia, 409–416. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.265

[6] Matthew Bradbury, Arshad Jhumka, and Matthew Leeke. 2018. Hybrid Online Protocols for Source Location Privacy in Wireless Sensor Networks. *J. Parallel and Distrib. Comput.* 115 (May 2018), 67–81. https://doi.org/10.1016/j.jpdc.2018.01.006

[7] Matthew Bradbury, Arshad Jhumka, and Carsten Maple. 2019. The Impact of Decreasing Transmit Power Levels on FlockLab To Achieve a Sparse Network. In *Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things (CPS-IoTBench '19)*. ACM, New York, NY, USA, 7–12. https://doi.org/10.1145/3312480.3313171

[8] Robert Bridson. 2007. Fast Poisson Disk Sampling in Arbitrary Dimensions. In *ACM SIGGRAPH 2007 Sketches (SIGGRAPH '07)*. ACM, San Diego, CA, USA, Article 22, 1 pages. https://doi.org/10.1145/1278780.1278807

[9] Michael I. Brownfield, Theresa Nelson, Scott Midkiff, and Nathaniel J. Davis. 2010. Wireless Sensor Network Radio Power Management and Simulation Models. *The Open Electrical & Electronic Engineering Journal* 4 (2010), 21–31.

[10] Zhichao Cao, Daibo Liu, Jiliang Wang, and Xiaolong Zheng. 2017. Chase: Taming Concurrent Broadcast for Flooding in Asynchronous Duty Cycle Networks. *IEEE/ACM Transactions on Networking* 25, 5 (Oct 2017), 2872–2885. https://doi.org/10.1109/TNET.2017.2712671

[11] Ricardo C. Carrano, Diego Passos, Luiz C. S. Magalhaes, and Célio V. N. Albuquerque. 2014. Survey and Taxonomy of Duty Cycling Mechanisms in Wireless Sensor Networks. *IEEE Communications Surveys Tutorials* 16, 1 (First 2014), 181–194. https://doi.org/10.1109/SURV.2013.052213.00116

[12] Mauro Conti, Jeroen Willemsen, and Bruno Crispo. 2013. Providing Source Location Privacy in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys and Tutorials* 15, 3 (2013), 1238–1280. https://doi.org/10.1109/SURV.2013.011413.00118

[13] Mianxiong Dong, Kaoru Ota, and Anfeng Liu. 2015. Preserving Source-Location Privacy through Redundant Fog Loop for Wireless Sensor Networks. In *13th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*. IEEE, Liverpool, UK, 1835–1842. https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.274

[14] Adam Dunkels. 2011. *The ContikiMAC Radio Duty Cycling Protocol.* Technical Report. Swedish Institute of Computer Science. http://dunkels.com/adam/dunkels11contikimac.pdf SICS Technical Report T2011:13.

[15] Simon Duquennoy, Beshr Al Nahas, Olaf Landsiedel, and Thomas Watteyne. 2015. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys '15)*. ACM, New York, NY, USA, 337–350. https://doi.org/10.1145/2809695.2809714

[16] Atis Elsts. 2019. LogisticLoss radio medium. https://github.com/contiki-ng/cooja/pull/15 Accessed: 2020-07-04.

[17] Guangjie Han, Hao Wang, Jinfang Jiang, Wenbo Zhang, and Sammy Chan. 2018. CASLP: A Confused Arc-Based Source Location Privacy Protection Scheme in WSNs for IoT. *IEEE Communications Magazine* 56, 9 (Sep. 2018), 42–47. https://doi.org/10.1109/MCOM.2018.1701062

[18] Guangjie Han, Hao Wang, Xu Miao, Li Liu, Jinfang Jiang, and Yan Peng. 2020. A Dynamic Multipath Scheme for Protecting Source-Location Privacy Using Multiple Sinks in WSNs Intended for IIoT. *IEEE Transactions on Industrial Informatics* 16, 8 (2020), 5527–5538. https://doi.org/10.1109/TII.2019.2953937

[19] Jie Hao, Baoxian Zhang, and Hussein T. Mouftah. 2012. Routing protocols for duty cycled wireless sensor networks: A survey. *IEEE Communications Magazine* 50, 12 (December 2012), 116–123. https://doi.org/10.1109/MCOM.2012.6384460

[20] Wendi B. Heinzelman, Anantha P. Chandrakasan, and Hari Balakrishnan. 2002. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on* 1, 4 (Oct. 2002), 660–670. https://doi.org/10.1109/TWC.2002.804190

[21] Zhen Hong, Rui Wang, Shouling Ji, and Raheem Beyah. 2018. Attacker Location Evaluation-based Fake Source Scheduling for Source Location Privacy in Cyber-Physical Systems. *IEEE Transactions on Information Forensics and Security* 14, 5 (2018), 1337–1350. https://doi.org/10.1109/TIFS.2018.2876839

[22] Timofei Istomin. 2015. LPL channel probing is too short. https://github.com/tinyos/tinyos-main/issues/338.

[23] Arshad Jhumka and Matthew Bradbury. 2017. Deconstructing Source Location Privacy-aware Routing Protocols. In *Proceedings of the Symposium on Applied Computing (SAC'17)*. ACM, Marrakech, Morocco, 431–436. https://doi.org/10.1145/3019612.3019655

[24] Arshad Jhumka, Matthew Bradbury, and Matthew Leeke. 2015. Fake source-based source location privacy in wireless sensor networks. *Concurrency and Computation: Practice and Experience* 27, 12 (2015), 2999–3020. https://doi.org/10.1002/cpe.3242

[25] Pandurang Kamat, Yanyong. Zhang, Wade Trappe, and Celal Ozturk. 2005. Enhancing Source-Location Privacy in Sensor Network Routing. In *25$^{th}$ IEEE International Conference on Distributed Computing Systems (ICDCS'05)*. IEEE, Columbus, OH, USA, 599–608. https://doi.org/10.1109/ICDCS.2005.31

[26] Jack Kirton, Matthew Bradbury, and Arshad Jhumka. 2017. Source Location Privacy-Aware Data Aggregation Scheduling for Wireless Sensor Networks. In *37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Atlanta, GA, USA, 2200–2205. https://doi.org/10.1109/ICDCS.2017.171

[27] Jack Kirton, Matthew Bradbury, and Arshad Jhumka. 2018. Towards optimal source location privacy-aware TDMA schedules in wireless sensor networks. *Computer Networks* 146 (2018), 125–137. https://doi.org/10.1016/j.comnet.2018.09.010

[28] Philip Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. 2005. *TinyOS: An Operating System for Sensor Networks.* Springer Berlin Heidelberg, Berlin, Heidelberg, 115–148. https://doi.org/10.1007/3-540-27139-2_7

[29] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. 2013. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *International Conference on Information Processing in Sensor Networks (IPSN)*. ACM/IEEE, Philadelphia, PA, USA, 153–165. https://doi.org/10.1145/2461381.2461402

[30] Jun Long, Mianxiong Dong, Kaoru Ota, and Anfeng Liu. 2014. Achieving Source Location Privacy and Network Lifetime Maximization Through Tree-Based Diversionary Routing in Wireless Sensor Networks. *IEEE Access* 2 (2014), 633–651. https://doi.org/10.1109/ACCESS.2014.2332817

[31] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. 2002. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1$^{st}$ ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*. ACM, Atlanta, Georgia, USA, 88–97. https://doi.org/10.1145/570738.570751

[32] Raja Manjula and Raja Datta. 2018. A novel source location privacy preservation technique to achieve enhanced privacy and network lifetime in WSNs. *Pervasive and Mobile Computing* 44 (2018), 58 – 73. https://doi.org/10.1016/j.pmcj.2018.01.006

[33] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. 2004. The Flooding Time Synchronization Protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*. ACM, Baltimore, MD, USA, 39–49. https://doi.org/10.1145/1031495.1031501

[34] Kiran Mehta, Donggang Liu, and Matthew Wright. 2012. Protecting Location Privacy in Sensor Networks against a Global Eavesdropper. *IEEE Transactions on Mobile Computing* 11, 2 (Feb. 2012), 320–336. https://doi.org/10.1109/TMC.

2011.32

[35] David Moss, Jonathan Hui, and Kevin Klues. 2007. *Low Power Listening*. Technical Report. TinyOS Core Working Group. btnode.ethz.ch/static_docs/tinyos-2.x/pdf/tep105.pdf TEP 105.

[36] Lilian C. Mutalemwa and Seokjoo Shin. 2019. Achieving Source Location Privacy Protection in Monitoring Wireless Sensor Networks through Proxy Node Routing. *Sensors* 19, 5 (2019), 1037. https://doi.org/10.3390/s19051037

[37] Ali Nassiri, M. A. Razzaque, and Abdul Hanan Abdullah. 2016. Isolated Adversary Zone for source location privacy in Wireless Sensor Networks. In *International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, Paphos, Cyprus, 108–113. https://doi.org/10.1109/IWCMC.2016.7577042

[38] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. 2006. Cross-Level Sensor Network Simulation with COOJA. In *Proceedings of 31st Conference on Local Computer Networks*. IEEE, Tampa, FL, USA, 641–648. https://doi.org/10.1109/LCN.2006.322172

[39] Celal Ozturk, Yanyong Zhang, and Wade Trappe. 2004. Source-location privacy in energy-constrained sensor network routing. In *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks (SASN '04)*. ACM, Washington DC, USA, 88–93. https://doi.org/10.1145/1029102.1029117

[40] Alejandro Proaño, Loukas Lazos, and Marwan Krunz. 2017. Traffic Decorrelation Techniques for Countering a Global Eavesdropper in WSNs. *IEEE Trans. on Mobile Computing* 16, 3 (March 2017), 857–871. https://doi.org/10.1109/TMC.2016.2573304

[41] Mayank Raj, Na Li, Donggang Liu, Matthew Wright, and Sajal K. Das. 2014. Using data mules to preserve source location privacy in Wireless Sensor Networks. *Pervasive and Mobile Computing* 11 (2014), 244–260. https://doi.org/10.1016/j.pmcj.2012.10.002

[42] Mo Sha, Gregory Hackmann, and Chenyang Lu. 2013. Energy-efficient Low Power Listening for wireless sensor networks in noisy environments. In *Proceedings of the 12th International Conference on Information Processing in Sensor Networks (IPSN '13)*. ACM/IEEE, Philadelphia, Pennsylvania, USA, 277–288. https://doi.org/10.1145/2461381.2461415

[43] Arafat A. A. Shabaneh, Azizi Mohd Ali, Chee Kyun Ng, Nor Kamariah Noordin, Aduwati Sali, and Mohd. Hanif Yaacob. 2014. Review of Energy Conservation Using Duty Cycling Schemes for IEEE 802.15.4 Wireless Sensor Network (WSN). *Wireless Personal Communications* 77, 1 (01 Jul 2014), 589–604. https://doi.org/10.1007/s11277-013-1524-y

[44] Min Shao, Wenhui Hu, Sencun Zhu, Guohong Cao, S. Krishnamurth, and T. La Porta. 2009. Cross-layer Enhanced Source Location Privacy in Sensor Networks. In *6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '09)*. IEEE, Rome, Italy, 1–9. https://doi.org/10.1109/SAHCN.2009.5168923

[45] Petros Spachos, Dimitris Toumpakaris, and Dimitrios Hatzinakos. 2014. Angle-Based Dynamic Routing Scheme for Source Location Privacy in Wireless Sensor Networks. In *79th Vehicular Technology Conference (VTC Spring)*. IEEE, Seoul, South Korea, 1–5. https://doi.org/10.1109/VTCSpring.2014.7022833

[46] Yanjun Sun, Omer Gurewitz, and David B. Johnson. 2008. RI-MAC: A Receiver-initiated Asynchronous Duty Cycle MAC Protocol for Dynamic Traffic Loads in Wireless Sensor Networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*. ACM, Raleigh, NC, USA, 1–14. https://doi.org/10.1145/1460412.1460414

[47] Wei Tan, Ke Xu, and Dan Wang. 2014. An Anti-Tracking Source-Location Privacy Protection Protocol in WSNs Based on Path Extension. *Internet of Things Journal, IEEE* 1, 5 (Oct. 2014), 461–471. https://doi.org/10.1109/JIOT.2014.2346813

[48] D. Tang, J. Gu, Y. Yu, Y. Yang, W. Han, and X. Ma. 2018. Source-location Privacy Based On Dynamic Mix-ring In Wireless Sensor Networks. In *2018 International Conference On Computing, Networking And Communications (ICNC)*. IEEE, Maui, HI, USA, 327–331. https://doi.org/10.1109/ICCNC.2018.8390237

[49] Na Wang, Junsong Fu, Jiwen Zeng, and Bharat K. Bhargava. 2018. Source-Location Privacy Full Protection in Wireless Sensor Networks. *Information Sciences* 444 (2018), 105–121. https://doi.org/10.1016/j.ins.2018.02.064

[50] Yi Yang, Min Shao, Sencun Zhu, and Guohong Cao. 2013. Towards Statistically Strong Source Anonymity for Sensor Networks. *ACM Trans. Sen. Netw.* 9, 3, Article 34 (June 2013), 23 pages. https://doi.org/10.1145/2480730.2480737

[51] Lin Yao, Lin Kang, Fangyu Deng, Jing Deng, and Guowei Wu. 2015. Protecting source–location privacy based on multirings in wireless sensor networks. *Concurrency and Computation: Practice and Experience* 27, 15 (2015), 3863–3876. https://doi.org/10.1002/cpe.3075

[52] Qian Zhou, Xiaolin Qin, and Xiaojun Xie. 2018. Hiding Contextual Information for Defending a Global Attacker. *IEEE Access* 6 (2018), 51735–51747. https://doi.org/10.1109/ACCESS.2018.2869947