**Manuscript version: Author’s Accepted Manuscript**
The version presented in WRAP is the author’s accepted manuscript and may differ from the published version or Version of Record.

**Persistent WRAP URL:**
https://wrap.warwick.ac.uk/177113

**How to cite:**
Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

**Copyright and reuse:**
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

**Publisher’s statement:**
Please refer to the repository item page, publisher’s statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.
To Checkpoint or Not to Checkpoint: That is the Question

Double Blind
do not reveal authors

Abstract
One of the major shortcomings in IoT/sensor networks is the finite energy supply available for computation and communication. To circumvent this issue, energy harvesting has been proposed to enable embedded devices to mitigate their dependency on traditional battery-driven power source. However, energy supply due to energy harvesting often varies, leading to nodes crashing due to energy exhaustion, with application(s) losing their state. Efficient state checkpointing in non-volatile memory (NVM) has been proposed to enable forward progress, albeit at the expense of significant overhead (viz., energy and time). In this paper, we show that, for a certain class of applications, state checkpointing may adversely affect the performance of the applications. This is different to checkpointing in traditional distributed system where network topology is generally assumed to be stable.

1 Introduction
As the popularity of smart applications such as smart cities, smart homes increases, so will the number of Internet of Things (IoT) devices (or nodes). In fact, it is projected that the number of connected IoT devices in 2025 will reach 27 billion, up from 12 billion in 2021 [19]. Such IoT devices are typically resource-constrained, i.e., they have a weak CPU, low memory and finite energy source. However, the problem of a finite energy source is proving to be a limiting factor in the adoption of large-scale IoT networks, whereby a node will crash and lose its state when it runs out of energy. To circumvent this problem, new techniques are required to reduce the reliance on finite sources. In essence, there is now a drive to develop IoT devices with energy sources that can provide energy reliably for longer, possibly indefinitely.

Energy harvesting is now becoming one of the most commonly utilised solutions to circumvent the finite energy problem. The energy sources harvested can be ambient, mechanical, human, bioenergy and hybrid [23]. These sources, in short, provide bursts of energy over a longer duration of time. However, energy harvesting is generally highly variable across space and time [8], which can often limit the intended pervasiveness of such IoT networks. Capacitors, as energy storage devices, can help with smoother energy supply. However, they often dominate the IoT devices in virtue of their sizes. As such, a node will eventually crash due to energy exhaustion, losing all of its state. The node will then recover when it has harvested enough energy. Thus, periods of normal computation and periods of energy harvesting become interleaved unpredictably which, in turn, impact computation both in terms of energy and time.

Checkpointing has been proposed to overcome this major problem. In short, checkpointing is the process of (periodically) capturing a snapshot of the application or system state and saving it on stable (non-volatile) memory (NVM) [15]. After a crash, when the system has lost its state, the system reloads the last saved state from NVM. While checkpointing on NVM allows application state to persist across failures, they induce a non-negligible overhead on the system; for example, when flash memory is used as NVM, the energy cost is orders of magnitude larger than most system operations[8], meaning that checkpointing has to be used carefully. While the use of FRAM improves these figures, checkpoints often represent the dominating factor in an application’s energy and time profile [9].

Checkpointing is a well-known technique in distributed systems and high performance computing that has been used to tolerate failures[15, 16, 18] and persist application state across failures. With the advances in IoT device hardware, many of them are now equipped with NVM such as flash or FRAM, making checkpointing a viable technique to persist state when a node crashes due to energy exhaustion. However, most works on checkpointing in IoT domains have focused on centralised applications and there is a dearth of work that focuses on checkpointing IoT networks applications.

Most works, if not all, on message logging and checkpointing in distributed systems will not work in IoT networks due to the very different nature of the applications and requirements.

Problem Differently to checkpointing centralised application, an additional requirement when checkpointing in a dis-
Checkpointing is used to ensure forward recovery and needs to ensure state consistency in distributed systems. It is the process of periodically (or other) storing a state on non-volatile storage [15]. Literature on distributed systems differentiates the checkpointing mechanism into two classes: local state and global state. Local checkpoints preserve the state of local processes at such instants. In contrast, global checkpoints save the entire system state, which includes all local states and channel states[15, 16] while care should be taken to maintain certain properties such as causality.

In this section, we survey examples of checkpointing techniques and their optimisations in current state-of-the-art research. A fault recovery system to a checkpointed correct node data (local state) and node trust degree from permanent storage is proposed for wireless sensor nodes (WSN) [18]. This recovery architecture maintains network connectivity and improves node link quality during fault recovery. In addition, local incremental checkpointing is implemented on a transiently-powered node’s memory protection unit (MPU). This research shows that utilizing MPU hardware handler to a checkpoint is more efficient and reduces checkpointing overhead than using designed software [7].

On the other hand, the Sytare system is proposed to persist a node’s peripheral state by checkpointing it [5, 6]. The
focus is on checkpointing the node’s peripheral state, which
includes serial interface, ADC, timer or radio transceiver,
rather than checkpointing the node state. Also, a system
called HarvOS uses code instrumentation for checkpointing
[9]. This system is triggered at compile time while benefitting
from a control flow graph (CFG) to decide when to check-
point. A consistency-aware adaptive checkpointing scheme
also solves the problem of inconsistent volatile and non-
volatile memory logs [26]. The inconsistency issue between
volatile RAM and non-volatile RAM occurs when reloading
the RAM state after a fault and collating it with the check-
pointed state.

A recent paper [3] reviewed the current works in transiently-powered networks (TPN) state retention mecha-
nisms. It breaks down the state retention to peripheral
state, program state, persistent timer keeping and the exist-
ing checkpointing strategy to copy-if-change and copy-used.
Furthermore, the trigger of the checkpointing mechanism is
to be either proactive or reactive.

Differential checkpointing (DICE) is a compile-time sys-
tem that determines the difference between a checkpointed
state and a volatile memory state and then performs a check-
point if necessary [2]. DICE is evaluated by integrating it
with proactive HarvOS [9] and MementOS [21], as well as
reactive Hibernus [4] and copy-if-change [8]. Results indi-
cate that DICE reduces checkpoint frequency, thereby con-
serving energy and shortening checkpoint duration, thereby
increasing service availability. Our work will show that it is
possible to reduce the number of checkpoints taken for spe-
cific classes of applications.

Thus, we observe that the state-of-the-art in checkpoint-
ing in transiently-powered IoT networks is mostly limited
to local node-related state checkpoints rather than network-
related states. We investigate this problem further, by run-
ing experiments on a real testbed, to gather initial insights.

3 IoT Protocols

In this section, we briefly present the two protocols we
will use as case studies in this work.

3.1 RPL Routing Protocol

RPL is a prominent Low-Power and Lossy Networks (LLNs)
routing protocol. LLNs are classified based on resource
constraints, such as limited memory, processing power, and
energy availability, with a lossy communication link
between nodes. RPL is designed on the foundation of
the IPv6 network stack, as delineated by the Internet En-
engineering Task Force (IETF) under the nomenclature RFC
6550 [25].

RPL is a distance vector proactive routing protocol which
establishes links between nodes by calculating the direction
to their next hop and the distance cost based on such metrics
[17]. The construction of network topology serves various
objectives, depending on the intended network functional-
ity. The default objective function employed by RPL is cen-
tred on reducing the Estimated Transmission Count (ETX)
from any given node to a designated RPL root node. The
 topology established by RPL takes the form of a Destination-
Oriented Directed Acyclic Graph (DODAG) to the design-
nated root node. The setup and maintenance of the DODAG

is achieved through three main control messages, namely
DIO (DODAG Information Object), DIS (DODAG Informa-
tion Solicitation) and DAO (Destination Advertisement Ob-
ject). The DODAG structure should persist throughout
the network’s lifespan, and RPL utilises Trickle timers to con-
trol the rate at which these control messages are generated.
Trickle timers rely on the Trickle algorithm, which involves
disseminating new network information to all nodes aperi-
odically and minimizing the broadcasting rate when the net-
work is stable [17].

3.2 LEACH

LEACH (Low-Energy Adaptive Clustering Hierarchy) is
a self-organizing and adaptive clustering protocol for IoT
networks. In the network, the nodes organize themselves
into local clusters, which are sets of nodes, with one node
taking on the role of a cluster-head. In hierarchical routing,
all members of a cluster relay their messages to the cluster-
head which, in turn, aggregates the messages before forward-
ing the resulting message. As such, cluster-head uses more
energy than cluster members.

To even the energy usage, LEACH uses randomization to
distribute the energy load evenly among the nodes in the net-
work, thereby extending the lifetime of the network. Thus,
LEACH uses randomized rotation of the expensive cluster-
head role such that it rotates among the various nodes, in
order to extend the lifetime of every network node.

LEACH routing protocol consists of a predefined number
of rounds. Each round begins with the advertisement, clus-
ter formation, and data scheduling phases. Upon completion
of these setup procedures, data transmission initiates. Dur-
ing the advertisement phase, every node decides whether or
not to act as a cluster-head by selecting a random number
from a uniform distribution between 0 and 1. The decision is
based on whether the selected number is below a threshold
determined by the formula cited in reference [13].

4 Problem Statement

In this section, we explain the problems we tackle in this
paper and enunciate some hypotheses to capture the nature
of the problems.

Closer analysis of the IoT protocols such as RPL and
LEACH suggests that IoT protocols contain a dynamic phase
and a static phase. The static phase occurs when the network
is stable, where the specification for the given problem (e.g.,
clustering or routing) is being satisfied. On the other hand,
the dynamic phase is triggered when the network is changing
such that the specification may be violated (e.g., the current
shortest path is no longer) and the network needs to evolve
to satisfy the specification again (e.g., a new shortest path
needs to be computed).

Global checkpointing in a distributed system involves
capturing a state that is consistent and there are protocols
that exist to achieve this [10]. However, such protocols are
not likely to work well in an IoT network for several reasons,
two of which are: (i) the network is lossy and (ii) the network
is multihop.

When the saved state of a crashed node is reloaded, two
possibilities exist: either (i) that local state is compatible
(i.e., consistent) with the global state of the application or (ii)
that local state is inconsistent with the global state, in which case the application may perform “sub-optimally” as the application will need to handle the inconsistency. Since, in IoT networks, several protocols, such as RPL and LEACH, are based on 1- or 2-hop neighborhood interactions, we take the term global to mean the “1- or 2-hop neighbourhoods”. In general, we are going to say the k-hop neighbourhood of a node, based on the specification of the problem. For example, RPL is based on the 1-hop neighbourhood of a node whereas computing a collision-free TDMA schedule involves a node knowing information about its 2-hop neighbours.

**Hypothesis 1:** Checkpointing application state when the application is in a dynamic phase can adversely affect the performance of the application.

The intuition behind this hypothesis is that, when a node runs out of energy (i.e., crashes), the application may need to reconfigure. Checkpointing a state prior to that reconfiguration will mean that, when the node recovers and reloads its state, that state will no longer be consistent with the new application configuration.

We now derive an expression that captures the probability of reconfiguration with the checkpointing period, from the perspective of a given node n and its k-hop neighborhood, where k is problem specific.

Let X be the random variable “number of network structure reconfiguration per unit time” within n’s k-hop neighborhood and let the checkpointing period be P time units. Assume X follows a Poisson distribution with parameter \( \lambda \). The expected number of structure reconfigurations between two checkpoints \( C_1 \) and \( C_2 \) is \( \lambda P \). We then denote by R, the random variable “number of network structure reconfiguration per checkpoint period”.

For the checkpoint \( C_1 \) of a node n to still be consistent with its k-hop neighborhood in the period \([C_1 \ldots C_2]\), there should have been no network reconfiguration in that time period P. Thus, we wish to calculate the probability that \( Pr(R = 0) \), which is given by:

\[
Pr(R = 0) = \frac{e^{\lambda P} (\lambda P)^0}{0!} = e^{\lambda P - \lambda P} = e^{0}
\]

The above suggests that, when \( \lambda \) is high due to several nodes crashing due to energy exhaustion, the probability of no reconfiguration is very small, meaning that the checkpoint \( C_1 \) is very likely stale, hence may lead to neighbourhood inconsistency if reloaded, thereby impact on the performance of the application.

**Hypothesis 2:** Checkpointing application state when the application is in a static phase can improve the performance of the application.

From the above equation, we observe that, when the network is stable and \( \lambda \) is very small, the probability of no reconfiguration is negligible, meaning that the checkpoint \( C_1 \) of node n will very likely be consistent with n’s k-hop neighbourhood.

**Hypothesis 3:** The impact of checkpointing on the efficiency of an IoT application will vary according to the failure patterns.

From a node n’s perspective, the higher the rate of features (i.e., energy exhaustion) in its k-hop neighborhood, the higher the number of network reconfigurations needed that will affect the state of n. This means that the likelihood that a checkpoint of n will be stale is high, thereby impacting the efficiency of the application.

In the rest of the paper, we conduct a number of experiments using RPL and LEACH to evaluate the truthfulness of the three hypotheses.

## 5 Methodology

In this section, we explain the experimental and network setup in terms of network topology, network size, network protocol, hardware specification, testbed specification and OS used.

In the rest of the paper, we will consider RPL as the application with a dynamic phase due to the fact that the DODAG will keep being updated due to node crashes. On the other hand, we observe that LEACH is relatively stable, even in the presence of node crashes. A period of instability will only happen when either the clusterhead crashes or when the clusterhead is rotated. Given a network of size N and the number clusters be \( C \), with \( C < < N \), then the probability of a clusterhead crashing is \( C/N \), when a node fails, giving rise to a low probability of clusterhead failure.

### 5.1 Experimental Setup

The Contiki-NG was used as the booting operating system in the experiments. It is an open-source, secure, and reliable operating system for IoT devices [20]. This operating system is suitable for IoT networks, which are typically low-power and lossy networks (LLN). The experiments were conducted on the FIT-IoT Lab testbed. It is a large testbed with more than 1500 IoT sensor nodes scattered around France [1]. The nodes are deployed over a 1.20 m x 1.20 m grid topology. We used the Lille site for the deployment as it supports extensive multi-hop experiments.

Furthermore, we used the IoT-LAB M3 board on this testbed. This board comprises an STM32 (ARM Cortex M3) microcontroller, an ATMEL radio interface operating at 2.4 GHz, and four sensors. It also includes a 128-Mbits external NOR flash memory and a three-LED lighting system [1, 14]. The number of nodes used throughout the experiments is 100-200, subject to testbed resource controls, and each experiment lasted two hours. The experimental details were summarised in Table 1. The metrics that were extracted are (i) message delivery ratio, and (ii) energy consumption. As stated in Table 1, the failure percentage is 10%, 20% and 30%. So, this paper does not consider a higher failure percentage for network connectivity reasons, as a higher failure percentage will very likely cause a network partition. However, looking for a higher failure percentage and maintaining network connectivity will be considered in future work.

We simulate a transiently-powered network by crashing a node, then making the node to recover and then subsequently joining the network again [22]. The maximum number of crashed nodes at one point is 30% of the network size, with the lowest being 10%.

One of the checkpointing techniques proposed is accomplished by writing only changed variables onto non-volatile memory and subsequently reading from non-volatile mem-
network.

transient network characteristics, i.e., LEACH in a perfect

RPL: Dynamic Application

The dynamic application we consider is RPL, a routing

protocol for low-power and lossy networks, as stated previ-

ously. RPL uses several metrics, such as link quality, to de-

termine the parent of a node in the DODAG. Please observe

that in RPL, a node \( n \) is only directly affected by its 1-hop

neighborhood.

The application with a static phase we consider is LEACH

and we consider three variants: (i) implementing a ver-

sion of the LEACH routing protocol for low-power and

lossy networks, specifically the transiently-powered check-

pointed LEACH, which we denote as CP-LEACH. To eval-

uate the effectiveness of checkpointing, we compare CP-

LEACH with (ii) a transiently-powered version of LEACH

without checkpointing and (iii) with LEACH without any,

transient network characteristics, i.e., LEACH in a perfect

network.

We have developed two C functions for checkpointing

data: one to write changes to the NOR external flash and the

other to read data from the NOR external flash for invariant

checks. As before, unlike most current state-of-the-art, our

checkpointed data is related to the application state rather

than the CPU state of the node.

We checkpoint the data whenever a cluster-head is se-

lected, and the data includes the selected cluster-head for

each round. We assume nodes will crash and recover within

the same round, making the checkpointed data valid and

functional, as explained in Section 4. Please observe that

in LEACH, a node \( n \) is only directly affected by its 1-hop

neighborhood. Thus, a cluster-head information is local if a

node recovers in the same crashed round.

Our findings, which we detail in the next section, demon-

strate that checkpointing is necessary for LEACH (i.e., im-

proves the performance of LEACH), due to the fact that it is

a static application.

Experimental Results

The findings obtained from the experiments are detailed

in this section.

RPL: Dynamic Application

In this section, we present the results of our study on the

impact of checkpointing dynamic applications, specifically

the checkpointed version of Routing Protocol for Low-power

and Lossy Networks (CP-RPL). We evaluate the packet de-

livery ratio (PDR) of RPL, both with and without check-

pointing, to understand the impact of checkpointing on

network performance.

Random Failures: We conduct a 120-minute experiment,
during which sender nodes send messages to the sink with

Table 1. Experiments Setup

<table>
<thead>
<tr>
<th>Experiment Setup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Contiki-NG</td>
</tr>
<tr>
<td>Routing protocol</td>
<td>RPL Classic and LEACH</td>
</tr>
<tr>
<td>Network size</td>
<td>100-200 nodes</td>
</tr>
<tr>
<td>Experiment duration</td>
<td>2 hours</td>
</tr>
<tr>
<td>Testbed</td>
<td>Fit IoT Lab</td>
</tr>
<tr>
<td>Node type</td>
<td>IoT-LAB M3</td>
</tr>
<tr>
<td>Failure percentage</td>
<td>10%, 20%, 30%</td>
</tr>
<tr>
<td>Trickle [min, doubling]</td>
<td>[8,12] defaults [25]</td>
</tr>
</tbody>
</table>

This paper employs the default RPL Trickle timer param-

eters to regulate the transmission of DODAG Information

Object (DIO) control messages. Accordingly, the minimum

interval between two consecutive DIOs is set at 4 seconds,

while the maximum time interval is 17 minutes, per these

default values [25].

Our transiently-powered checkpointed RPL\(^1\) (denoted by

CP-RPL) implementation is evaluated by comparing it to a

transiently-powered RPL that does not use checkpointing.

\(^1\)We say transiently-powered RPL to mean RPL in a transiently-powered

network.
various sending periodicities of 60 seconds, 90 seconds, and 120 seconds. During each experiment, we randomly crash a given proportion of nodes, to mimic energy exhaustion. The PDR is calculated as the ratio of the total received packets at the destination node to the packets sent from the source nodes. In this paper, all nodes are source nodes. Our results show (see Figure 2) that the highest PDR is obtained for RPL (i.e., RPL without checkpointing), which approaches 94%.

However, when we checkpoint the state of RPL, i.e., when CP-RPL is considered, the Packet Delivery Ratio decreases significantly from 81% at the highest with a 10% crashing rate to 15% at the lowest, when the crashed proportion is 30%. These findings suggest that checkpointing an application in its dynamic has a negative impact of its PDR, in that it reduces the PDR for CP-RPL due to a node (re)use of network information that is no longer consistent with its 1-hop neighbourhood, leading to the loss of messages.

Furthermore, we observe that the negative impact of increasing crash percentages on the network PDR is more severe on CP-RPL than RPL. This is because the recovered node in RPL benefits from requesting new network information from the node neighbours rather than retrieving stale information from checkpointed RPL state.

In addition to PDR, we also measure the network energy consumption of RPL and CP-RPL. We have captured the energy monitoring profile provided by the FIT IoT-LAB and their energy calculation formula [12]. Consequently, a control node dedicated hardware installed on the FIT IoT-LAB node is used to measure the node energy consumption. The average energy consumption of all nodes over the course of the experiment is compared, and it is shown that checkpointing consumes more energy, as expected. Our findings demonstrate (see Figure 3) that CP-RPL consumes more energy than RPL. The reason why energy consumption for RPL decreases with time is due the number of active nodes left in the network over time. Overall, the impact of checkpointing on RPL is two-fold: (i) reduced PDR and (ii) increased energy. This is in contradiction to previous works, such as [8], which showed the benefits of checkpointing. These observations support our first hypothesis enunciated in Section 4, that checkpointing an IoT application when it’s in its dynamic phase can adversely affect its performance.

Impact of Application Message Periodicity: We now study the effect of application message periodicity on the network PDR in the presence of transient power failure for CP-RPL. The results, depicted in Figure 4, suggest that, at a 30% crash rate, the PDR is not impacted by the transmission periodicity due to the high level of dynamism in the DODAG, which can be due to several reasons, e.g., high churn, DODAG poisoning among others due to non-viable parents. We also conjecture that the high number of control messages generated can lead to nodes not updating their state as needed, resulting in stale (inconsistent) state, thereby causing a substantially lower PDR. Thus, we observe that checkpointing adversely affects the performance of CP-RPL.

However, there are relatively fewer re-configurations at 10% and 20% crashed rates than at 30%, meaning that, for applications with high periodicity, fewer number of messages will get lost. However, if the message frequency is high, a higher number of messages will be lost, as shown in Figure 4, resulting in lower PDR.

Figure 5 shows the energy consumption under different application message periodicity. It can be observed that, for a given failure rate and at high periodicity, the energy consumed is less due to less state changes, thereby requiring less checkpoints.

Predefined Failure Patterns: Additionally, we examine the
impact of spatial and temporal properties of crashes on the network PDR for RPL and CP-RPL. We look at two dimensions for each of spatial and temporal distribution, viz. clustered (C) and far (F). We denote by CS-CT failure to denote that the crash are clustered spatially (i.e., all crashed nodes are close to each other) and clustered temporally (i.e., the crashes occur very close to each other in time). On the other hand, FS-FT denotes crashed nodes that are spatially far from each other, i.e., the crashes are independent as well as being far temporally.

Figure 6 shows that, when node failures occur in close proximity both spatially and temporally (CS-CT), the PDR for RPL is reduced to 79%, while CP-RPL experiences an extremely low PDR of 13%. This is due to the high dynamism occurring in the 1-hop neighborhood of a given node affected by the failures. At the other end of the spectrum (FS-FT), the level of dynamism in RPL is less due to the fact that crashes are both spatially and temporally independent, meaning that not too many nodes are “simultaneously” crashing in a given neighborhood. From Figure 6, we can observe that the spatial dimension holds a greater impact as it induces a higher dynamism in a given neighborhood. On the other hand, when crashed nodes are “far apart”, the decrease in PDR is less.

Furthermore, Figure 7 illustrates that energy consumption is almost comparable for spatial and temporal crashes as the crash rate is constant at 20% for all experiments. The slightly higher energy consumption for nodes crashes that are temporally far apart can be attributed to the fact that, when the state changes, every such change is checkpointed, i.e, CS-FT has highest energy due to many checkpoints are recorded due to the dynamism in a neighborhood. On the other hand, it seems to be that, when crashes occur close to each other temporally, some changes may not be checkpointed due to the speed at which state changes are happening.

Figure 5. Random Failures: The Effect of Transmission Frequency on Energy Usage for CP-RPL

Figure 6. Failure Patterns: Impact of Spatial and Temporal Distribution of Failures on PDR for RPL and CP-RPL

Figure 7. Failure Patterns: Impact of Spatial and Temporal Distribution of Node Failures on Energy Usage on CP-RPL

Overall, our results for a network-based application such as RPL showed that the energy spent to perform state checkpointing to enable state to persist over node crashes is not only being wasted, but is also leading to worse performance.

Overall, the impact of spatial and temporal distribution of crashes and the proportion of crashes have varying impact on CP-RPL, thereby corroborating our hypothesis 3.

6.2 LEACH: Static Application

In this section, we present the results of our study into the impact of checkpointing on an application that is mostly static, specifically LEACH. We have implemented LEACH for Contiki-ng and we execute it in a network with no failures to obtain a baseline for PDR. We subsequently run LEACH in a transiently-powered network (LEACH-TPN) and a checkpointed version of LEACH in a transiently-powered network (CP-LEACH-TPN).

Figure 8 shows the PDR of LEACH under baseline conditions, where no node crashes occur, with a PDR of 95%. Additionally, Figure 8 illustrates the PDR of CP-LEACH-TPN and LEACH-TPN in the presence of a transiently-powered network (TPN). The results indicate that CP-LEACH-TPN exhibits better PDR than LEACH-TPN, with an average value of approximately 65%, whereas LEACH without checkpointing has a considerably lower PDR with an average value of around 33%, across various crash rates.

The information that nodes recorded was the information about their respective cluster heads. During stable phases, the probability of the head crashing is very low (as we argued before), meaning that this information is unlikely to be stale. Thus, upon recovery, a node can readily be linked to its clusterhead.
These observations corroborate the second hypothesis enunciated in Section 4, which stated that checkpointing a stable application will likely result in a higher PDR.

![Figure 8. The Impact of of a Transiently Powered Network (TPN) on PDR of the LEACH Algorithm and CP-LEACH](image)

7 Conclusion and Future Work

In this paper, the objective is to investigate the efficacy of checkpointing in transiently-powered IoT networks. We observed that IoT applications have a dynamic and stable phase. We ran a number of experiments and proposed three hypotheses, namely that (i) checkpointing an application in its dynamic phase is likely to negatively impact its performance, (ii) checkpointing an application in its static phase is likely to boost its performance and (iii) failure patterns impact the efficacy of checkpointing.

As such, our first hypothesis goes counter to works such as [8] as our results suggest that checkpointing is bad for dynamic applications. On the other hand, the results that support our second hypothesis also explain the reason for the correctness of the works in [8]: The applications the authors use are static (in the sense we previously define).

As future work, we are currently investigating necessary and sufficient conditions for when checkpointing is needed. We are also looking at the notion of adaptive checkpointing so that an application can predict whether the network will be stable so it can start checkpointing or, if the network is unstable, when checkpointing will not be required, i.e., checkpoint only when needed.

8 References


