To Checkpoint or Not to Checkpoint: That is the Question

Jawaher Alharbi*
University of Warwick, Coventry, UK
Jawaher.Alharbi@warwick.ac.uk

Arshad Jhumka†
University of Warwick, Coventry, UK
H.A.Jhumka@warwick.ac.uk

Abstract

One of the major shortcomings in Internet of Things (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 sources. Nevertheless, 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 overheads (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 or smart homes increases, so also, the number of Internet of Things (IoT) devices (or nodes) increases. In fact, it is projected that the number of connected IoT devices by 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 central processing unit (CPU), low memory and finite energy source. Indeed, the problem of a finite energy source is proving to be such a limiting factor in the adoption of large-scale IoT networks, due to the fact that a node tends to 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 or hybrid [23]. These sources, in short, provide bursts of energy over a longer duration of time. Nevertheless, 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 as a solution to overcoming 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, of an order of magnitude, larger than most system operations[8], meaning that checkpointing has to be used carefully. While the use of ferroelectric random-access memory (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 states across failures. With 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 check-

*Department of Information Technology, College of Computer, Qassim University, Buraydah, Saudi Arabia. Email:j.Harbi@qu.edu.sa

†Also with the School of Computing, University of Leeds, UK

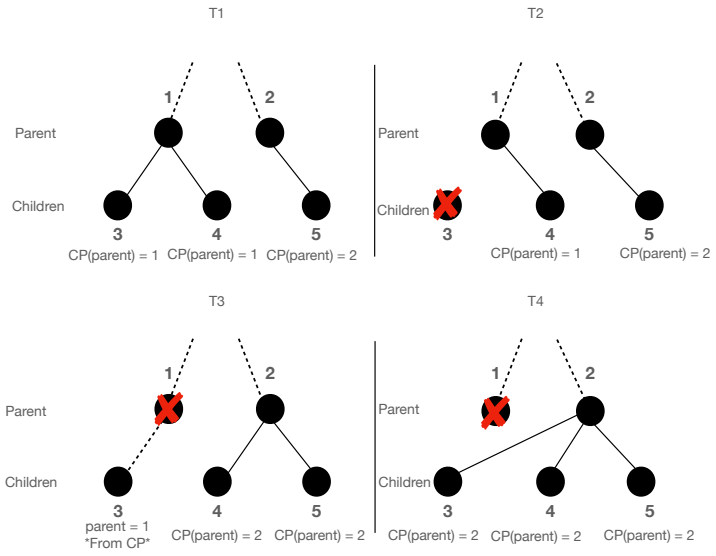

Figure 1. Checkpointing Issues in IoT Networks

pointing 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 distributed setting is that of ensuring the consistency of the (global) checkpoint. Also, given the dynamism of IoT networks, the states held at nodes are consistently being updated. As such, when data is retrieved from NVM after a crash, the possibility of the overall programme state becoming inconsistent is high. We explain this in more detail, based on Figure 1 that captures a RPL DODAG at a given time $T1$. RPL is a distance-vector routing protocol that is suitable for IoT networks, that establishes a Destination-Oriented Directed Acyclic Graph (DODAG) upon execution.

At time $T1$, assume that part of the DODAG is as shown in Figure 1. At this point, nodes $3,\ldots,5$ have checkpointed their states, including the i. d. of their respective parents. We illustrate some issues by focusing on the parent variable. At a given time $T2 > T1$, node 3 crashes, losing all of its volatile state, but with its parent set to 1 stored in its NVM. Before 3 recovers (due to energy harvesting), node 1 (node 3's parent) crashes. Upon recovery at $T3$, node 3 reloads its state from NVM and establishes node 1 as its parent. Until node 3 rejoins a new DODAG, its messages (and any messages sent to it) will be lost. At $T4$, node 3 changes its parent to node 2, thereby correcting the problem. Thus, there is the fact that node 3 checkpointed a state that, upon reload, was inconsistent with the network state that was the root cause of the problem.

As such, one important question arises: Does checkpointing have a negative impact on the performance of network applications in IoT networks? We investigate this problem from two different viewpoints, namely:

- Does network dynamism affect the efficiency of checkpointing?
- Does the failure pattern affect the efficiency of checkpointing?

We chose two applications namely (i) RPL, a standardised routing protocol for IoT networks and (ii) LEACH, a clustering algorithm that allows for hierarchical routing, to study the above effects.

Contributions

We make the following contributions:

- We study the two applications and observe that such applications can be split into two components: (i) a dynamic phase and (ii) a static phase.
- We run FIT-IoT testbed experiments on both applications under a range of failure scenarios.
- Our results show that (i) checkpointing during the dynamic stage impacts the packet delivery ratio significantly and (ii) checkpointing during the static stage boosts the packet delivery ratio significantly.

In effect, we observe that, for RPL, checkpointing does not appear to be needed, in that checkpointing may even have a negative impact on the performance of the application. In our initial work, we observe that checkpointing gains can be significant (i.e., no checkpointing is required), compared to previous works: e.g., whereas techniques such as DICE [2] checkpoints the differentials between states, thereby reducing the number of checkpoints. Our results show that checkpoints may not be needed for some applications, which differ from previous known results.

The structure of this paper is as follows: we discuss related work in Section 2. Following that, we introduce the used case studies in Section 3. We then explained the problem statement in Section 4. In Section 5, we describe the experimental setup and methodology used for evaluation. We then present the results in Section 6. We conclude the paper in Section 7.

2 Related Work

Checkpointing is used to ensure forward recovery and needs to ensure state consistency in distributed systems. It is the process of periodically (or otherwise) storing a state on non-volatile storage [15]. Literature on distributed systems differentiate the checkpointing mechanisms 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 utilising MPU hardware handler

for a checkpoint is more efficient and reduces checkpointing overheads rather 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 compilation time while benefiting from a control flow graph (CFG) to decide when to checkpoint. 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 checkpointed state.

A recent paper [3] reviewed the current works in transiently-powered networks (TPN) state retention mechanisms. It breaks down the state retention to peripheral state, programme state, persistent timer keeping and the existing 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 system that determines the difference between a checkpointed state and a volatile memory state and then performs a checkpoint 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 indicate that DICE reduces checkpoint frequency, thereby conserving energy and shortening checkpoint duration, thereby increasing service availability. Our work shows that it is possible to reduce the number of checkpoints taken for specific classes of applications.

we observe that the state-of-the-art in checkpointing 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 running experiments on a real testbed, to gather initial insights.

3 IoT Protocols

In this section, we briefly present the two protocols which 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 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 functionality. The default objective function employed by RPL is centred 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 designated root node. The setup and maintenance of the DODAG is achieved through three main control messages, namely DIO (DODAG Information Object), DIS (DODAG Information Solicitation) and DAO (Destination Advertisement Object). The DODAG structure should persist throughout the network's lifespan, and RPL utilises Trickle timers to control 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 aperiodically and minimising the broadcasting rate when the network 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 organise 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 forwarding the resulting message. As such, cluster-head uses more energy than cluster members.

To even the energy usage, LEACH uses randomisation to distribute the energy load evenly among the nodes in the network, thereby extending the lifetime of the network. Thus, LEACH uses randomised 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, cluster formation and data scheduling phases. Upon completion of these setup procedures, data transmission initiates. During 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 neighbourhood 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 is 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 λ. The expected number of structure reconfigurations between two checkpoints C_1 and C_2 is $P\lambda$. 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 $[C_1 \dots C_2]$, 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:

$$Pr(R=0) = \frac{exp^{(-P\lambda)}(P\lambda)^0}{0!}$$
$$= exp^{(-P\lambda)}$$

The above suggests that, when λ 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 this may lead to neighbourhood inconsistency if reloaded, thereby impacting 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 λ 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 failures (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 remainder 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 operating system (OS) used.

In the remainder of the paper, we 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 of clusters as 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 was 100-200, subject to testbed resource controls, and each experiment lasted two hours. The experimental details are 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 recover and then, subsequently, joining the network again [22]. The maximum number of

Table 1. Experiments Setup

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

crashed nodes at one point was 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 memory (NVM) when needed [8]. There are different types of non-volatile memory with different capabilities in terms of access speed, data retention, writing energy consumption and wear out. For instance, flash, MRAM, FRAM and others were used [11, 24]. IoT checkpointing state-of-the-art uses flash, and the current IoT testbed uses nodes that support flash as a non-volatile memory chip. As a result, we used flash as NVM, and we took advantage of the IoT-LAB M3 board's 128-Mbit external NOR flash to store the checkpoints. The Contiki-NG file system (CFS) API handles writing and reading from the external flash. It makes use of the coffee system functionality, which converts an expensive and complicated bit toggling to micro logging on flash memories and non-volatile ROM (EEPROM).

The accepted benchmark is two widely used routing protocols for the lossy and low-power networks, RPL and LEACH. In the following two sections, we present the instrumentation of the chosen two application protocols to understand the impact of checkpointing on their performance.

5.2 RPL: Application with Dynamic Phase

The dynamic application we consider is RPL, a routing protocol for low-power and lossy networks, as stated previously. RPL uses several metrics, such as link quality, to determine 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.

Contiki-ng provides two RPL protocol implementations: RPL-Classic and RPL-Lite. RPL-Lite, as the name implies, simplifies routing implementation by supporting a non-storing mode, increasing ROM availability and allowing only one DAG and one RPL instance to run at a time [25]. Subsequently, we chose RPL-Classic as the tested protocol because we wanted to capture and checkpoint data that RPL-Lite does not provide.

This paper employs the default RPL Trickle timer parameters 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. In this paper, unlike most current state-of-the-art approaches, the checkpointed data are related to the application state rather the state of the CPU of a node.

The checkpointing is executed by programming two C functions, one for writing data to the NOR external flash upon any changes and the other for reading data from the NOR external flash when an invariant check is required. The checkpointed data are (i) RPL dag information, (ii) node parent information and (iii) other. This data is checkpointed by any network node whenever such data changes. Node failures, node join and rejoin, inconsistent trickle timers, or a node's parent change could all cause these changes.

Our results show that RPL (with network variables) does not need checkpointing even when nodes fail due to energy exhaustion, thereby corroborating our hypothesis 1.

5.3 LEACH: Application with Static Phase

The application with a static phase we consider is LEACH and we consider three variants: (i) implementing a version of the LEACH routing protocol for low-power and lossy networks, specifically the transiently-powered checkpointed LEACH, which we denote as CP-LEACH. To evaluate 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 approaches, 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 selected, 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, demonstrate that checkpointing is necessary for LEACH (i.e., improves the performance of LEACH), due to the fact that it is a static application.

6 Experimental Results

The findings obtained from the experiments are detailed in this section.

6.1 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

[1] We say transiently-powered RPL to mean RPL in a transiently-powered network.

and Lossy Networks (CP-RPL). We evaluate the packet delivery ratio (PDR) of RPL, both with and without checkpointing, 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 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 state has a negative impact on 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 to 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 is 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) states, thereby causing a substantially

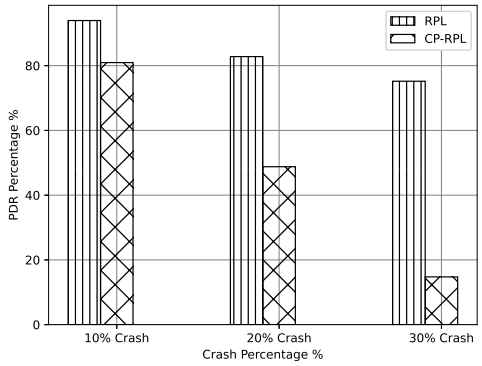

Figure 2. Random Failures: The Impact of Crash Proportion on the PDR of RPL and CP-RPL

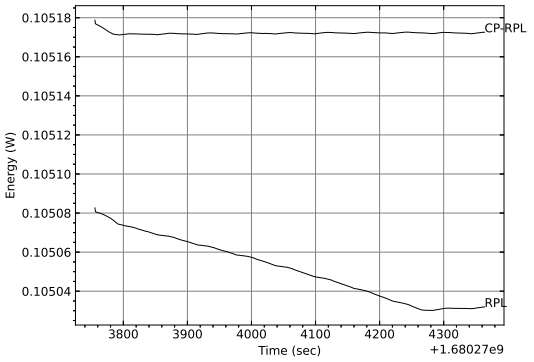

Figure 3. Random Failures: The Effect of CP-RPL and RPL on Energy Usage

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 numbers 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 periodicities. 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 utilise CS-CT failure to denote that the crashes 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

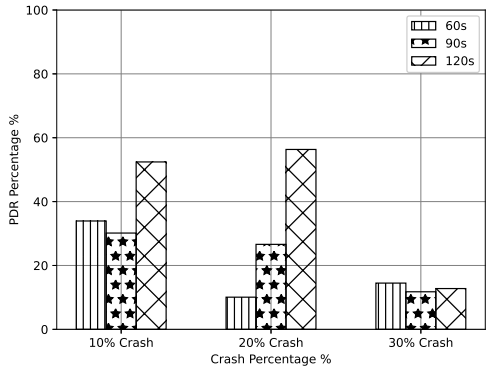

Figure 4. Random Failures: The Impact of Transmission Periodicity on the CP-RPL

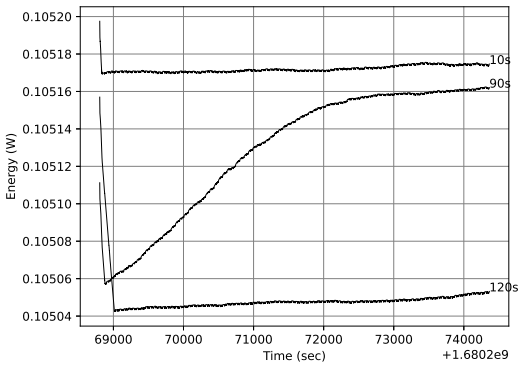

Figure 5. Random Failures: The Effect of Transmission Frequency on Energy Usage for CP-RPL

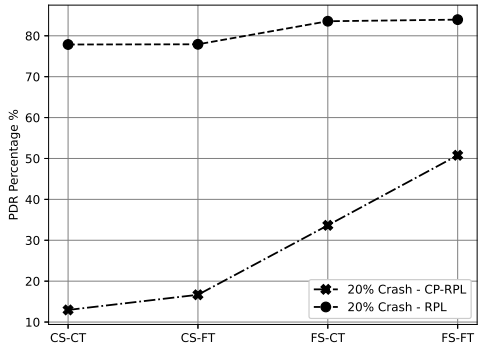

Figure 6. Failure Patterns: Impact of Spatial and Temporal Distribtution of Failures on PDR for RPL and CP-RPL

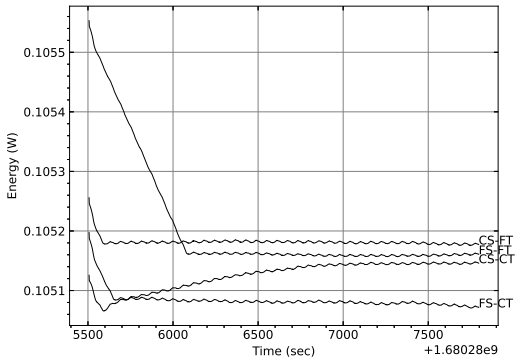

Figure 7. Failure Patterns: Impact of Spatial and Temporal Distribution of Node Failures on Energy Usage on CP-RPL

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 *n* 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 neighbourhood. 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 being recorded due to the dynamism in a neighbourhood. On the other hand, it seems to be the case 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.

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

Overall, the impact of spatial and temporal distribution of crashes and the proportion of crashes have varying impacts 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 ar-

gued 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.

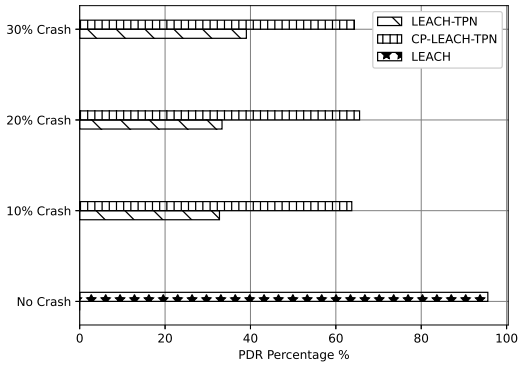

Figure 8. The Impact of of a Transiently Powered Network (TPN) on PDR of the LEACH Algorithm and CP-LEACH

7 Conclusion and Future Work

In this paper, the objective was 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 less effective 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 defined).

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

[1] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, et al. Fit iot-lab: A large scale open experimental iot testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE, 2015.

[2] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. Fast and energy-efficient state checkpointing for intermittent computing. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(6):1–27, 2020.

[3] S. Ahmed, N. A. Bhatti, M. Brachmann, and M. H. Alizai. A survey on program-state retention for transiently-powered systems. *Journal of Systems Architecture*, 115:102013, 2021.

[4] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, 2014.

[5] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. Peripheral state persistence for transiently-powered systems. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[6] G. Berthou, T. Delizy, K. Marquet, T. Risset, and G. Salagnac. Sytare: A lightweight kernel for nvram-based transiently-powered systems. *IEEE Transactions on Computers*, 68(9):1390–1403, 2018.

[7] G. Berthou, K. Marquet, T. Risset, and G. Salagnac. Mpu-based incremental checkpointing for transiently-powered systems. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 89–96. IEEE, 2020.

[8] N. Bhatti and L. Mottola. Efficient state retention for transiently-powered embedded sensing. In *International Conference on Embedded Wireless Systems and Networks*, pages 137–148, 2016.

[9] N. A. Bhatti and L. Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 209–220. IEEE, 2017.

[10] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985.

[11] T. Eshita, T. Tamura, and Y. Arimoto. Ferroelectric random access memory (fram) devices. In *Advances in non-volatile memory and storage technology*, pages 434–454. Elsevier, 2014.

[12] FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. Consumption monitoring. Accessed Jun. 12, 2022 [Online], 2020.

[13] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd annual Hawaii international conference on system sciences*, pages 10–pp. IEEE, 2000.

[14] HiKoB. Iot-lab node m3, 2013.

[15] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on software Engineering*, (1):23–31, 1987.

[16] A. D. Kshemkalyani and M. Singhal. *Distributed computing: principles, algorithms, and systems*. Cambridge University Press, 2011.

[17] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The trickle algorithm. Technical report, 2011.

[18] S. Mitra and A. Das. Distributed fault tolerant architecture for wireless sensor network. *Informatica*, 41(1), 2017.

[19] H. Mohammad. State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally, 2022. (18,05,2022).

[20] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes. The contiki-ng open source operating system for next generation iot devices. *SoftwareX*, 18:101089, 2022.

[21] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. In *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pages 159–170, 2011.

[22] D. Richardson, A. Jhumka, and L. Mottola. Protocol transformation for transiently powered wireless sensor networks. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1112–1121, 2021.

[23] T. Sanislav, G. D. Mois, S. Zeadally, and S. C. Folea. Energy harvesting techniques for internet of things (iot). *IEEE Access*, 9:39530–39549, 2021.

[24] R. C. Sousa and I. L. Prejbeanu. Non-volatile magnetic random access memories (mram). *Comptes Rendus Physique*, 6(9):1013–1021, 2005.

[25] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, and R. Alexander. Rpl: Ipv6 routing protocol for low-power and lossy networks. Technical report, 2012.

[26] M. Xie, C. Pan, M. Zhao, Y. Liu, C. J. Xue, and J. Hu. Avoiding data inconsistency in energy harvesting powered embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(3):1–25, 2018.