RelIoT: Reliability Simulator for IoT Networks

Kazim Ergun[1(✉)], Xiaofan Yu[1], Nitish Nagesh[2], Ludmila Cherkasova[3], Pietro Mercati[4], Raid Ayoub[4], and Tajana Rosing[1]

[1] University of California San Diego, La Jolla, USA
{kergun,x1yu,tajana}@ucsd.edu
[2] Technical University of Munich, Munich, Germany
nitish.nagesh@tum.de
[3] Arm Research, San Jose, USA
lucy.cherkasova@arm.com
[4] Intel Corporation, Hillsboro, USA
{pietro.mercati,raid.ayoub}@intel.com

Abstract. The next era of the Internet of Things (IoT) calls for a large-scale deployment of edge devices to meet the growing demands of applications such as smart cities, smart grids, and environmental monitoring. From low-power sensors to multi-core platforms, IoT devices are prone to failures due to the reliability degradation of electronic circuits, batteries, and other components. As the network of heterogeneous devices expands, maintenance costs due to system failures become unmanageable, making reliability a major concern. Prior work has shown the importance of automated reliability management for meeting lifetime goals for individual devices. However, state-of-the-art network simulators do not provide reliability modeling capabilities for IoT networks.

In this paper, we present an integrated reliability framework for IoT networks based on the ns-3 simulator. The lack of such tools restrained researchers from doing reliability-oriented analysis, exploration, and predictions early in the design cycle. Our contribution facilitates this, which can lead to the design of new network reliability management strategies. The proposed framework, besides reliability, incorporates three other interrelated models - power, performance, and temperature - which are required to model reliability. We validate our framework on a mesh network with ten heterogeneous devices, of three different types. We demonstrate that the models accurately capture the power, temperature, and reliability dynamics of real networks. We finally simulate and analyze two examples of energy-optimized and reliability-optimized network configurations to show how the framework offers an opportunity for researchers to explore trade-offs between energy and reliability in IoT networks.

1 Introduction

The Internet of Things (IoT) is a growing network of heterogeneous devices, combining residential, commercial, and industrial domains. Devices range from low-power sensors with limited computational capabilities to multi-core platforms

W. Song et al. (Eds.): ICIOT 2020, LNCS 12405, pp. 63–81, 2020.
https://doi.org/10.1007/978-3-030-59615-6_5

on the high-end. From small scale (e.g., smart homes) to large scale (e.g., smart cities) applications, the IoT provides infrastructure and services for enhancing the quality of life and use of resources. By 2025, the IoT is expected to connect 41 billion devices [14].

The unprecedented scale and heterogeneity of the IoT pose major research challenges that have not been faced before. While ongoing research efforts aim at optimizing power efficiency and performance [29,30], an aspect that has often been neglected is the reliability of the devices in the network. The common property for these devices is that they age, degrade and eventually require maintenance in the form of repair, component replacement, or complete device replacement. Since an enormous number of heterogeneous devices are interconnected in IoT networks, the maintenance costs will increase accordingly. Cisco recently anticipated that for 100K devices that operate IoT smart homes, around $6.7M/year will be spent for administration and technical diagnosis related to system failures, comprising between 30% to 70% of total costs [7]. Without a proper reliability management strategy, IoT solutions are strongly limited as it becomes infeasible to maintain increasingly large networks.

Exploring reliability management strategies requires a convenient tool for reliability evaluation. In respect of this, simulators are widely used tools in research and industry to evaluate and validate networks, to study novel methods without the need for real deployments when resources are limited. However, existing network simulators do not support aging/degradation and reliability modeling or analysis. Popular network simulators, e.g., ns-3 [6], OMNeT++ [5], and OPNET [9] are equipped with a rich collection of communication models, allowing the assessment of network performance (throughput, delay, utilization, etc.) under different protocols. Recent research also integrated energy models and an energy-harvesting framework to the platform [25,27]. Yet it is not possible to analyze the reliability of IoT networks with the existing tools because they lack built-in reliability models. It should be noted that we refer to the aging and reliability degradation of the hardware of an IoT device, and not to the communication reliability.

To address this gap in reliability analysis, we propose a simulation framework named *RelIoT*, which allows practical and large-scale reliability evaluation of IoT networks. The framework is implemented in ns-3 [6], a discrete-event network simulator with low computational overhead and low memory demands. Up to one billion nodes can be simulated with ns-3 [19]. In recent years, with the addition of models for various network settings and protocols through open-source contributions, ns-3 has established itself as a de facto standard network simulation tool. To allow reliability simulation in ns-3, *RelIoT* integrates the following modules:

- *Power Module:* Supports power consumption simulation for various workloads with configurable power models.
- *Performance Module:* Works in cooperation with the power module to provide performance predictions for a given workload.

- *Temperature Module:* Estimates the internal temperature of a device based on its power consumption.
- *Reliability Module:* Evaluates the device reliability using the existing thermal-based degradation models [16,23,32].

To the best of our knowledge, *RelIoT*[1] is the first reliability analysis framework for heterogeneous IoT networks, taking thermal characteristics as well as power and performance into account. *RelIoT* enables researchers to explore trade-offs between power, performance, and reliability of network devices. Moreover, *RelIoT* incurs only a marginal performance overhead on the default ns-3, making it scalable for simulating large networks. For the scalability analysis of the default ns-3, the reader is referred to previous works [19,20]. We validate our framework with two real-world experiments, showing *RelIoT* estimates power, performance, and temperature with errors of less than 3.8%, 4.5%, and $\pm 1.5\,^{\circ}\mathrm{C}$ respectively. We validate reliability models against the results from existing literature. Finally, we built a mesh network testbed to illustrate that *RelIoT* can effectively capture the average long-term power and thermal behavior of devices in a dynamic network. Finally, we provide example simulation results from *RelIoT* to motivate the need for reliability-aware management and to show the differences between energy-driven and reliability-driven management strategies.

The rest of the paper is organized as follows: Sect. 2 reviews power, performance, and reliability simulation techniques introduced by previous works. The overall structure of our proposed framework and details of models is elaborated in Sect. 3 and Sect. 4. Section 5 describes the evaluation setup and further discusses the results. The paper concludes in Sect. 6.

2 Related Work

Network Simulators for Power and Performance. Network simulators are used to study the behavior of computer networks and evaluate communication protocols prior to deployment. Popular examples are: ns-3 [6], OMNeT++ [5], and OPNET [9], all of which are discrete event-based and open-source. The standard versions of these network simulators are designed only for analyzing communication performance, lacking consideration for computation power, performance, and reliability of network devices.

Motivated by energy constraints in battery-powered sensor networks, several works have integrated power modeling and analysis with different granularity. Wu *et al.* [27] first introduced energy source models and device energy models to ns-3. They used existing analytical battery models and relied on hardware datasheets to build WiFi radio energy models. In another work named PASES [18], the authors construct accurate power consumption models for both processor and radio components of network devices by hardware design space

[1] *RelIoT* is available at: https://github.com/UCSD-SEELab/RelIoT.

exploration. In network simulators, usually, the accuracy of power models is compromised for attaining low computational costs. To provide flexible options for heterogeneous devices in an IoT network, *RelIoT* offers two configurable power models with different granularity, while allowing extensions for user-specified models.

For IoT performance, iFogSim [13] and EdgeCloudSim [22] incorporated computation performance (e.g. processing delay) to simulate end-to-end latency of a multi-level IoT structure including cloud servers, gateways, and sensors. Both toolkits employ low accuracy estimations such as look-up tables for latency and power analysis. *RelIoT* does a finer-grained estimation for different types of IoT devices running various applications.

Reliability Modeling and Management. Prior work has studied reliability degradation phenomena on processor-based systems. The considered failure mechanisms include Time-Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI) and Electromigration (EM), which all limit device lifetime [16,23,32]. In these works, the reliability degradation problem is approached in two steps: (i) Physical-level models are built to quantify the reliability degradation due to voltage and temperature stress, which are influenced by the environmental conditions and workload variations. (ii) Based on the reliability degradation models, a management algorithm is designed to optimize performance while satisfying reliability constraints. The trade-off between performance and reliability could be adjusted during runtime by voltage scaling [16,23,32], task scheduling [10], or both [17]. The recent work by Mercati *et al.* [17] implements the above-mentioned models on a mobile phone, showing as much as a one-year improvement on lifetime with dynamic reliability management. Despite the impressive results on individual devices, reliability management for IoT networks is yet to be investigated. Recently, a dynamic optimization approach was proposed to manage battery reliability degradation in IoT networks, but their work does not consider the reliability of other device components (e.g., processor) [12].

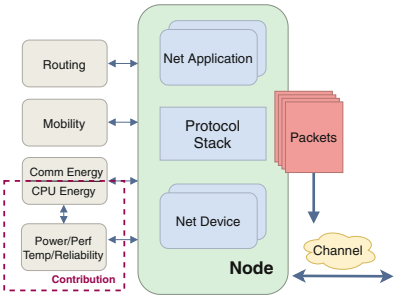
In this work, we propose *RelIoT*, a framework for end-to-end reliability simulation in IoT networks to enable investigation of reliability trade-offs and prototyping of reliability management strategies. We develop and integrate power, performance, temperature, and reliability modules into ns-3. In contrast to prior work on network simulators, *RelIoT* offers temperature and reliability estimation for the networked devices.

3 Reliability Framework for RelIoT

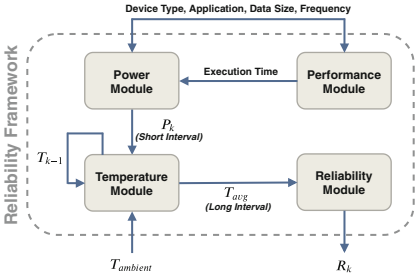
In this section, we give a background on ns-3 and its features, then describe the overall structure of *RelIoT* and its integration with ns-3.

3.1 ns-3 Preliminary

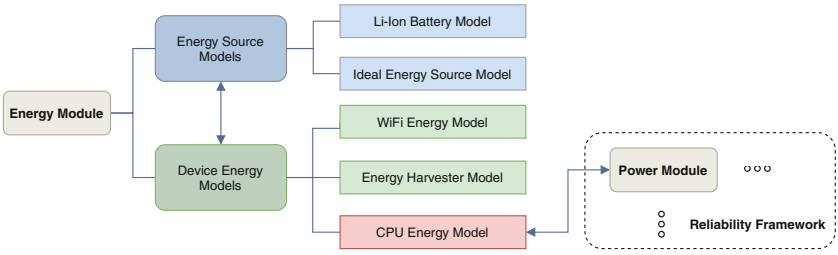
ns-3 is built as a system of libraries that work together to simulate a computer network. To do simulation using ns-3, the user writes a *C++* program that links the various elements from the library needed to describe the communication network being simulated. ns-3 has a library of *objects* for all of the various elements that comprise a network (*objects* are highlighted in italics). *Nodes* are a representation of computing devices that connect to a network. Sensors, routers, hubs, gateways, and servers in the IoT architecture can be all considered *Node* objects. Figure 1a shows the structure of a typical ns-3 *Node*. *Net Devices* represent the physical device that connects a *Node* to a communications *Channel*. For example, the *Net Device* can be a simple Ethernet network interface card or a wireless IEEE 802.11 device, and the *Channel* could be a fiber–optic link or the wireless spectrum. *Packets* are the fundamental unit of information exchange in a network. A *Packet* contains headers describing the information needed by the protocol implementation and a payload which represents the actual data being communicated between network devices. Each protocol in the *Protocol Stack* performs some operation on network packets and then passes them to another layer in the stack for additional processing. The *Net Applications* are simple networking applications that specify the attributes of communication policies between devices. All of these individual components are aggregated on the *Node* objects



(a) ns-3 node structure



(b) Proposed reliability framework



(c) Energy module and power module interconnection

Fig. 1. ns-3 integration of the reliability framework

to give them communication ability and set up networking activity. Other modules, such as *Routing*, *Mobility*, and *Energy*, can be installed to provide additional functionality to *Nodes*.

3.2 Overview of the Proposed Framework

Our proposed framework consists of separate modules for power, performance, temperature, and reliability, as shown in Fig. 1b. IoT devices can run some applications to process the sensed or collected data before sending it to a central entity. In this case, as soon as an application starts, the performance module first calculates its execution time. Then, the power module gives an estimate of power consumption within the execution interval of the application. If the IoT device is not running any applications, then idle power consumption is estimated. Given the power estimation, ambient temperature, the temperature module outputs an estimated temperature, which is fed to the reliability module. Finally, reliability is calculated based on temperature. The modules operate on two different time scales: *Long Intervals*, on the order of days that it takes for reliability to change, and *Short Intervals*, on the order of milliseconds. Both performance and power values are updated every *Short Interval*. Reliability estimation is computationally expensive, so it is only done once every *Long Interval* using the average temperature over each interval. The underlying mechanisms of each module are be explained in Sect. 4.

3.3 Integration with ns-3

As shown in Fig. 1a, our framework is implemented as an additional set of modules that can be aggregated on the *Nodes*, adhering to the structure and conventions of ns-3. The power and performance modules provide functions to other modules for querying power consumption and execution time values. We also provide an interface connecting the ns-3 energy module and our reliability framework (Fig. 1c). The energy module (proposed in [27]) consists of a set of energy sources and device energy models. An *EnergySourceModel* is an abstraction for the power supply (e.g. battery) of a *Node*. The *DeviceEnergyModel*s represent energy consuming components of a *Node*, for example, a WiFi radio. We implement a model called *CPUEnergyModel* as a child class of *DeviceEnergyModel*. The features of *CPUEnergyModel* are as follows:

- It is designed to be state-based, where the CPU can take the states *Idle* or *Busy*. The CPU will be *Busy* while processing packets received, e.g., while executing some applications such as encryption, decryption, compression, or Machine Learning (ML) algorithms.
- To determine when a transition occurs between states, a *PhyListener* is used. In ns-3, *PhyListener* is an object that monitors the network packet transmissions and receptions at the physical (PHY) layer. After an *Idle* node completes receiving data of specified size, the *PhyListener* notifies *CPUEnergyModel*. Then, the specified application is executed and state is set to *Busy*.

– It calculates the total energy consumed according to the power consumption value acquired from the power module and the execution time value acquired from the performance module. Then, it updates the remaining energy of the energy source described by *EnergySourceModel*.

4 Modules and Device Behavior Modeling

In this section, we describe the functionality of the proposed modules and interfaces and present underlying models in detail.

4.1 Power Module

The power module supports functions for running and terminating an application and switching between CPU states *Idle* and *Busy*. The value of power consumption is updated at a predefined period *Short Interval*, according to the selected power model. The power and temperature modules are interconnected; power consumption updates subsequently lead to temperature updates.

From cycle-accurate, instruction-level analysis to functional-level analysis, there are numerous power modeling techniques at different levels of abstraction. Low-level models use a fine-grain representation of the CPU, which usually implies that the time required for power estimation is large due to high computational complexity. This is undesirable for network simulations because it becomes very time consuming to simulate networks with a great number of nodes. In our framework, we offer two CPU power models having low model complexity while still providing good estimation accuracy. To improve the extensibility of the simulator for custom applications, we have included functionality for users to add new models to the power module. Parameters of the power models are configurable through external interfaces.

Frequency & Utilization-Based Power Model. The idea of estimating CPU power consumption on embedded devices based on CPU frequency and utilization is well studied. In a previous work [31], the authors use a linear combination of frequency and utilization to characterize the CPU power of a smartphone, achieving less than 2.5% average error. Similarly, we use linear models in our simulator to predict CPU power consumption P_{CPU}. The equation is given as:

$$P_{CPU}(t) = a \cdot f(t) + b \cdot u(t) + c \tag{1}$$

where $f(t)$ and $u(t)$ are CPU frequency and utilization at time t respectively. The coefficients a, b, c are learned through linear regression based on datasets collected on real devices. The frequency & utilization-based power model provides a good estimation accuracy for CPU power estimation on embedded devices. However, it requires frequency and utilization traces as inputs to the simulator which might not be available in practice.

Application-Based Power Model. The power consumption of embedded devices varies depending on the running application. An application-based model is convenient when there are only high-level functional properties are available, e.g.

input data size. Different applications have different power trends (i.e. linear, exponential, etc.) as the size of input data increases. Furthermore, the power consumed by running the same applications varies for different devices. In our framework, we adopt the modeling methodology proposed in [11], where the authors characterize and verify power models of running ML algorithms on edge devices (i.e. Raspberry Pi) and servers. They train, test, and cross-validate four regression models (linear, polynomial, log, and exponential regression), and select the best one. The input is the size of processed data by the application and the output is power consumption for these models. We leverage this methodology to deploy models for Raspberry Pi's and servers, but also apply the same methodology to build our own models for microcontrollers such as Arduinos. In addition to the 22 ML algorithms modeled in [11], our framework delivers a CPU power model for Multilayer Perceptron (MLP) based on the number of MAC (multiply-accumulate) operations. The same modeling approach can be applied to other neural network architectures such as Convolutional Neural Networks (CNNs).

4.2 Performance Module

IoT systems usually need to satisfy some performance requirements to provide adequate Quality of Service (QoS). To evaluate and monitor the performance of deployed applications and hence the overall network, we implement a performance module. Various metrics can be used to quantify performance, e.g., throughput, response time, etc. The performance metric is application-specific. For example, delay and throughput are critical in multimedia streaming applications whereas information accuracy is the main criterion for performance in some ML applications.

In our current release, we provide an *Execution Time Model*. We use the input data size of the application or number of MAC operations it needs to perform to estimate the application execution time. To build the model, we measure the execution times of various applications on a target device, then fit regression models to the collected data. Certain performance metrics can be calculated using the execution time value. For example, let t_{exec} be the execution time of an application, then its throughput can be obtained as D/t_{exec} where D is the input data size. In addition, end-to-end delay of a network path can be computed as the sum of communication and computation delays among the path (communication delay can be obtained using default ns-3 modules).

For both the power and performance modules, users are able to configure coefficients of the existing model or add new models with provided APIs.

4.3 Temperature Module

The goal of the temperature module is to estimate CPU temperature (based on CPU power consumption and ambient temperature) and to calculate the average temperature over a *Long Interval*. We adopt a thermal modeling strategy that can be used for any IoT device.

We assume that we do not have knowledge about the information describing topological and physical parameters of the device (e.g., we do not know material characteristics and layers of the devices' PCB board) so we cannot do a physical simulation of the process. To have an acceptable level of complexity in our simulator, we work on high-level information gathered from the coarse-grained thermal sensors of the device's key heat sources. Such information is available in most of the devices today like smartphones and single-board computers (e.g., Raspberry Pi).

Let the number of the heat sources be n and let $T_k \in \mathbb{R}^n$ represent the vector of temperatures observed by thermal sensors and $P_k \in \mathbb{R}^n$ be the power consumed by the heat sources at time instant k. Each heat source is assumed to have one thermal sensor measuring its temperature. Then, temperature T_{k+1} at time instant $k+1$ can be predicted given the current temperature T_k and power P_k at time k. The discrete-time state-space model of the device's thermal behavior is expressed in Eq. (2) [8].

$$T_{k+1} = A \cdot T_k + B \cdot P_k + C \cdot T_k^{env} \tag{2}$$

where $A, B \in \mathbb{R}^{n \times n}$ are defined as the state and the input matrices. T_k^{env} is the ambient temperature and C is a vector of coefficients which weighs the impact of ambient temperature on each heat source's internal temperature. We use system identification methods to derive the model from measured power and temperature traces. A, B and C parameters are different for each class of devices, so we offer multiple device thermal models and made the parameters configurable through the temperature module API. The order of the model is equal to the number of the heat sources n. In our initial work, we use $n = 1$, where the only source is CPU. However, the extension to multiple sources is straightforward in our framework. For example, if a power model for GPU is provided, then power consumption values from both CPU and GPU can be used to predict temperature.

The temperature module updates the states in Eq. (2) at a time resolution of *Short Interval*, the same time granularity as power estimation updates. On the other hand, average temperature $\bar{T}$ is calculated for every *Long Interval* denoted *LI*. $\bar{T}$ is the exponential moving average of past temperature values in the interval k to $k + LI$.

$$\bar{T}_{k+1} = \alpha \cdot T_k - (1 - \alpha) \cdot \bar{T}_{LI} \tag{3}$$

where α is a weighing coefficient that is configured depending on the length of interval LI.

4.4 Reliability Module

The reliability module is the last component in the power, temperature, reliability module hierarchy. Temperature is estimated using power, while reliability is estimated using temperature. Unlike power and temperature, reliability is a slowly changing variable. Therefore, reliability can be estimated on a longer

time scale, on the order of hours or days. Reliability degradation is affected more by average stress over a long time interval rather than instantaneous stress. We leverage these properties to calculate reliability sparsely because reliability models are highly compute-intensive. The reliability module does estimation every *Long Interval*, using temperatures averaged over the interval. It polls the temperature module to fetch the average temperature $\bar{T}_{LI}$ every LI, then $\bar{T}_{LI}$ is reset to start a new averaging operation.

Reliability is defined as the probability of not having failures before a given time t. To obtain the overall reliability of a processor, the effects of different failure mechanisms should be combined. We use the sum-of-failure-rates model as in RAMP [23], which states that the processor is a series failure system; the first instance of a failure due to any mechanism causes the entire processor to fail. In our reliability model, the single device reliability is a product of the reliabilities due to different failure mechanisms such as Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), Electromigration (EM) and Thermal Cycling (TC). These mechanisms all depend on thermals.

Time Dependent Dielectric Breakdown (TDDB) Reliability Model. The thin gate oxide layer in transistors introduces a risk of breakdown and shortening devices lifetime. Due to gate oxide degradation, which is a non-reversible mechanism with a cumulatively increasing impact, a breakdown occurs. The reliability of a single transistor i with oxide thickness x_i subject to oxide degradation can be expressed as [24]:

$$R_i(t) = e^{-a(\frac{t}{\gamma})^{\beta x_i}} \tag{4}$$

where t is the time-to-breakdown, a is the device area normalized with respect to the minimum area, and γ and β are respectively the scale parameter and shape parameter. The scale parameter γ represents the characteristic life, which is the time where 63.2% of devices fail, and it depends on voltage and temperature. The shape parameter β, instead, is a function of the critical defect density, which in turn depends on oxide thickness, temperature and applied voltage. $R(t)$ is a monotonically decreasing function with values in the range of [0, 1] indicating the probability that the system will not fail.

The reliability of the entire chip R_C can be expressed as the product of single transistor reliabilities:

$$R_C(t) = \prod_{i=1}^{m} R_i(t) = e^{\sum_{i=1}^{m} -a_i(\frac{t}{\gamma_i})^{\beta_i x_i}} \tag{5}$$

m is the number of transistors on the chip which can be on the order of millions. Since different regions of the chip have similar temperatures, the complexity possessed by large m on the computation of Eq. (5) can be reduced by assuming the same scale and shape parameters over the chip [32].

The R_C expression in Eq. (5) assumes a constant temperature applied from time $t = 0$, thus it is only representative of static systems. To capture the dynamics of reliability under varying temperature, we discretize the time and

calculate reliability at each time step as shown in Eq. (6). The temperature is assumed to be constant between discrete time steps.

$$R_k = R_{k-1} - \Big(R_C(t_{k-1}, T_{k-1,k}) - R_C(t_k, T_{k-1,k})\Big) \tag{6}$$

In Eq. (6), k indicates the k^{th} time instant and $T_{k-1,k}$ is the temperature experienced by the chip between the time instants $k-1$ and k. We set this interval between adjacent time steps as the *Long Interval* and let $T_{k-1,k}$ be equal to the average temperature $\bar{T}_{LI}$ of the corresponding *LI*.

The reliability module can work with any failure mechanism or combination of multiple mechanisms as long as the mechanism can be described by a function $R_C(t)$, as in Eq. (5). For example, the module can be extended to include NBTI and HCI if we describe the reliability functions associated with these mechanisms, respectively R_{NBTI} and R_{HCI}. Then, by the sum-of-failure-rates approach, the reliability module calculates the total system reliability as the product of the functions associated with the single mechanisms as $R_C(t) = R_{TDDB}(t) \cdot R_{NBTI}(t) \cdot R_{HCI}(t)$. Equation (6) would not need any modifications since it is general and does not depend on a specific $R_C(t)$.

5 Experiments and Results

In this section, we first present validation results on a three-node network topology, comparing power, performance, and temperature measurements from experiments with the simulated traces. We then use a testbed with a mesh network of 10 heterogeneous nodes to evaluate the accuracy of the simulator under different networking conditions and temperatures. We cannot explicitly validate reliability because it requires long term experiments and specialized degradation sensors. Finally, we illustrate how the proposed simulator is useful in exploring energy, performance, reliability trade-offs in a network and show that it can be used to implement reliability-aware strategies. We analyze examples of energy-optimized and reliability-optimized network configurations to motivate reliability-aware network design and management.

5.1 Validation and Evaluation

Three-Node Network Topology. To validate the device models and to verify the functionality of the simulator modules, we use a simple three-node network. The setup consists of an ESP8266 WiFi microchip with microcontroller, a Raspberry Pi 3 (RPi3), and a PC. The devices communicate over WiFi (IEEE 802.11b) and transmit/receive TCP/IP packets using MQTT protocol [4]. The ESP8266 samples random data as a sensor node, runs median filtering to preprocess the data, and sends filtered data to RPi3. The data is further processed by an application on the RPi3, or the computation can be offloaded to the PC. This type of computation offloading is common in IoT edge devices and is representative of their usual operation [21]. If the application is chosen to be offloaded, then the RPi3 is only responsible of relaying incoming data to the PC.

In our three-node experiments, we collected 5 different measurements synchronously:

(i) RPi3 power consumption (via HIOKI 3334 power meter [2]),
(ii) RPi3 CPU temperature (via built-in temperature sensor),
(iii) ESP8266 power consumption (via INA219 power monitor [3]),
(iv) ESP8266 CPU temperature (via built-in temperature sensor),
(v) Ambient temperature (via DHT22 temperature sensor [1]).

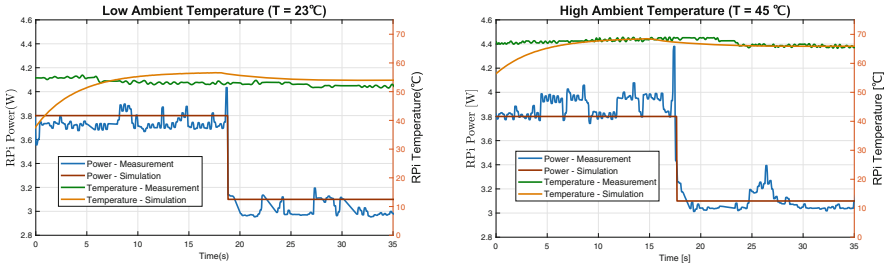


Fig. 2. RPi3 power and temperature traces

Measurements and simulation results are presented in Fig. 2 for an example test case under two different ambient temperatures. The goal here is to show a temporal view of the simulator output, particularly in a dynamic case where the simulated device has a varying workload. In this experiment, the RPi3 runs a data processing application with incoming data input from ESP8266 for the first 15–20 s. After that, the application is offloaded to the PC and the RPi3 only relays data while its CPU is *idle*. As shown in Fig. 2, the simulator output follows the real power and temperature traces with a mean error of 3.42% and 6.19% in low ambient temperature, and with a mean error of 2.69% and 3.97% in high ambient temperature. The discrepancy between real and simulated temperatures at the beginning of each plot is because of the initial condition set for the temperature in the simulator. The temperature starts from a lower initial condition and reaches a steady-state value.

Overall, applying the same modeling methodology of reference [11], we estimate the execution time and energy consumption of the RPi3 for 23 different ML applications with average errors of 3.8% and 4.5%, respectively. For the CPU temperature, the state-space model predictions stays within ±1.5 °C of measurements at steady-state, for all applications.

Mesh Network Topology. To show that our simulator can correctly capture devices' behavior in a more complicated scenario, we simulate a larger network under different configurations and operating conditions, then validate it using our testbed. As shown in Fig. 3, the testbed spans a whole floor in UCSD CSE

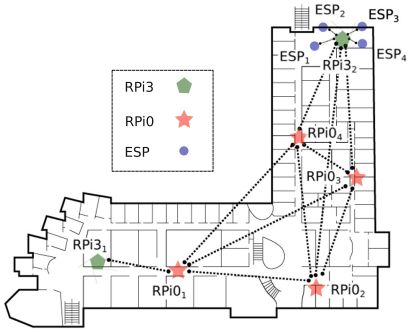


Fig. 3. Mesh network topology

department building, including two Raspberry Pi 3 (RPi3), four Raspberry Pi 0 (RPi0) and four ESP8266. Data is generated from each node and communicated to the sink node $RPi3_1$ in multiple hops via MQTT. The network of all RPis works in an ad-hoc manner, while all ESP8266s forward their data to $RPi3_2$ that is a gateway for that local area. Not all devices can communicate with each other because some pairs are out of communication range. The connections are depicted in Fig. 3. Using this setup, we both implement and simulate following scenarios:

Scenario 1. $RPi3_1$ and $RPi0_1$ process the data, while the other devices only communicate. The ambient temperature for each network device is approximately 25 °C.

Scenario 2. The same devices process data as Scenario 1. We use a heater that raises the ambient temperature around $RPi0_1$ to 37 °C, while the rest of the devices are in the normal ambient condition of 25 °C.

Scenario 3. The data processing duties of $RPi3_1$ and $RPi0_1$ are distributed between $RPi0_2$, $RPi0_3$, and $RPi0_4$. Therefore, each of these three devices only transmits the outputs of data processing tasks to $RPi0_1$, which directly forwards them to $RPi3_1$. $RPi0_1$ is still in a heated environment of 37 °C.

An ML application can be split and distributed to edge devices, which allows us to realize the different configurations in these scenarios for allocating data processing without changing the overall application behavior [26,28]. Figure 4 illustrate the power and temperature distribution of the devices, while Fig. 5 shows the simulated reliability traces in a year. We only depict the measurement and simulation statistics on $RPi0s$ in each scenario, but the rest show similar trends. Comparing collected traces to simulation logs, our result shows that *RelIoT* is able to estimate average power within ±0.11 W (∼11%), and average temperature within ±4 °C (∼9%). It can be seen from Fig. 4 that, although extremities in both power and temperature are difficult to track, *RelIoT* is able to precisely capture the averages in different configurations. Scenario 3 distributes the workload to other $RPi0s$, thus significantly reduces the network traffic. Consequently, power and temperature of both $RPi0_1$ and the rest $RPi0s$ drop, which is also

reflected in the simulation *RelIoT* starts simulation from a device temperature of 35 °C, which explains why the minimum temperature of *RelIoT* consistently locates at 35 °C.

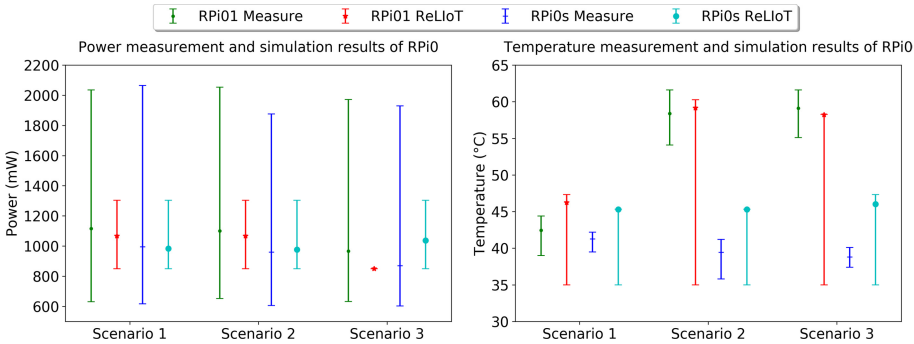


Fig. 4. Collected and simulated statistics of RPi0s (average, max, and min values).

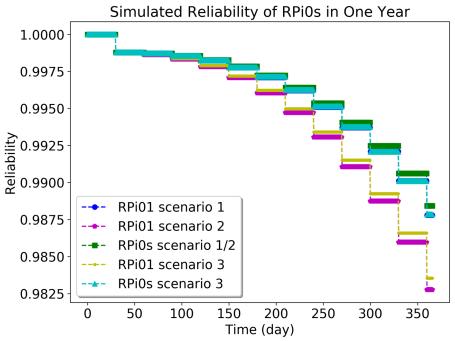


Fig. 5. Reliability degradation of RPi0s in one year.

The power and temperature validation experiments lasts 300 s, but we simulate the network for a time-span of one year to observe the long-term reliability changes. The stair pattern in Fig. 5 is a result of *RelIoT* updating the reliability by the end of each *Long Interval*. In all scenarios, reliability remains fairly high if the device is in normal ambient temperature. In Scenario 1, $RPi0_1$ degrades slightly faster than the rest of the $RPi0_s$ due to its data processing workload. However, in Scenario 2, the raised ambient temperature together with its workload lead to a drastic degradation in reliability. In such case, workload reallocation as in Scenario 3 can alleviate degradation. The network devices in environments with low temperatures can take on a higher workload to mitigate reliability problems of the quickly degrading devices. The result in Fig. 5 implies the necessity for reliability-aware management in IoT networks.

5.2 Reliability-Aware Management

Most of the IoT devices are battery-powered and/or rely on energy harvesters with limited energy sources. Therefore, traditionally, many network management solutions aim at optimizing the energy consumption while satisfying some Quality-of-Service (QoS) constraints (throughput, delay, jitter, network coverage, etc.). In this context, reliability is also a design parameter that can be optimized or a certain overall reliability constraint can be subjected to the network. Although correlated, the optimal energy efficiency and reliability usually are not ensured by the same management strategy. The designers need to find good trade-offs between energy savings and reliability. In this section, we show how our simulator addresses this issue by making reliability-aware management and design possible. To emphasize the differences between two approaches and to motivate reliability-aware management, we provide simulation results for different scenarios of energy-optimized and reliability-optimized network management strategies using the topology in Fig. 3.

Energy-Optimized. Our interest here is to partition an application into smaller tasks and find the task allocation that maximizes the lifetime of a network. Many ML applications can be partitioned while preserving functionality [26,28]. For each device in the network, the energy consumed for computing and communicating data of size s is given as:

$$\text{Computation:} \qquad P^{\xi}_{device}(s) \times t_{exec}(s) \tag{7}$$

$$\text{Communication:} \qquad P_{wifi}(d, BW) \times \frac{s}{BW} \tag{8}$$

where ξ denotes the application, t_{exec} is the application execution time, d denotes the communication distance, and BW is the communication bandwidth. $P_{wifi}(d, BW)$ is the power consumption of WiFi which can be parameterized by distance and bandwidth allocation BW [11]. In an energy-optimized application partition, the mapping of tasks to the devices depends on:

(i) Power characteristics of the application,
(ii) Execution time,
(iii) Allocated bandwidth,
(iv) Distance between the neighbouring devices.

We adopt the convex optimization formulation from [15] and apply it to our problem, with a slight modification by adding the computation energy term in Eq. (7). We find the optimal partitioning of the application such that the *maximum* energy consumption among network devices due to computation and communication of the data is minimized.

Reliability-Optimized. Similar to the previous case, we map the tasks of an application to the network devices. We use the same solution approach, but this time, the objective is to maximize the *minimum* reliability among network

devices. Reliability of each device is $R_{C,device}(t, T)$, which is dependent on time and temperature. We simulate a time horizon t_{sim}, so we want to optimize for $R_{C,device}(t_{sim}, T)$. This is under the assumption of environment temperature T_{amb} being constant for the entire horizon. In the following experiments, a static solution (constant for the whole time horizon) is simulated for both energy-optimized and reliability-optimized cases, but it can be made dynamic by solving for the current energy and reliability estimates at each time instant, as in (6). In this way, the solution can adapt to changing network configurations (bandwidth, applications) and operating conditions (environment temperature).

Figure 6 presents the comparison of energy-optimized and reliability-optimized solutions for different bandwidth and environment temperature configurations. The network devices run a part of a data processing application where the optimal partitions are determined by the two approaches. The energy-optimized partition brings 1.0%, 9.1%, and 10.9% better energy efficiency

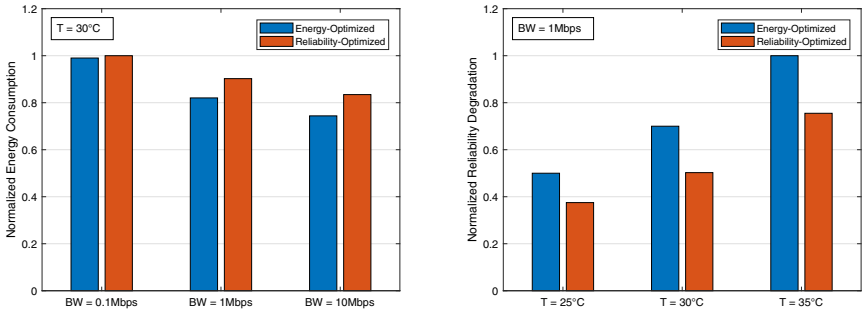


Fig. 6. Energy consumption and reliability degradation results for two approaches

compared to the reliability-optimized partition for 0.1 Mbps, 1 Mbps, and 10 Mbps bandwidth configurations respectively. Referring to Eq. (8), it can be seen that as bandwidth increases, the time it takes to communicate data of size s decreases, hence, decreasing the communication energy. The difference of energy efficiencies between two solutions are increasing with bandwidth because the energy-optimized solution leverages the decrease in communication energy and allocates more communication instead of computation to the higher energy consuming network devices. On the other hand, the reliability-optimized partition results in 25.0%, 28.2%, and 24.5% less reliability degradation compared to the energy-optimized partition for 25 °C, 30 °C, and 35 °C environment temperatures respectively. The reliability-optimized solution allocates less computation on the most degrading network devices, conserving reliability. These results show that, although being correlated, the optimal energy efficiency and reliability do not yield from the same management strategy. Therefore, if the concern is particularly the reliability, a reliability-aware management strategy should be adopted.

6 Conclusion

We presented a novel framework for the reliability analysis of IoT networks using the ns-3 network simulator. The proposed framework can be used to explore trade-offs between power, performance, and reliability of devices in a network. We validated our reliability framework in two experimental setups: a three-node network and a ten-node mesh network. Additionally, we motivated the need for reliability-aware management through example simulation results of energy-optimized and reliability-optimized management strategies. As future work, we plan to leverage our framework for design space exploration (DSE) of IoT networks. We can simulate, explore, and check the feasibility of different network configurations in terms of different objectives such as energy efficiency, reliability, and performance. We believe that our contribution will help researchers to study the reliability degradation problem in large-scale networks.

Acknowledgements. This work was supported in part by SRC task #2805.001, NSF grants #1911095, #1826967, #1730158 and #1527034, and by KACST.

References

1. DHT22 Datasheet. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf/
2. Hioki3334 Powermeter. https://www.hioki.com/en/products/detail/?product_key=5812
3. INA219 Datasheet. http://www.ti.com/lit/ds/symlink/ina219.pdf/
4. MQTT MQ Telemetry Transport. https://mqtt.org/
5. OMNeT++ Discrete Event Simulator. https://omnetpp.org/
6. The ns-3 Network Simulator. https://www.nsnam.org/
7. The Hidden Costs of Delivering IIoT (2016). https://www.cisco.com/c/dam/m/en_ca/never-better/manufacture/pdfs/hidden-costs-of-delivering-iiot-services-white-paper.pdf
8. Beneventi, F., Bartolini, A., Tilli, A., Benini, L.: An effective gray-box identification procedure for multicore thermal modeling. IEEE Trans. Comput. **63**(5), 1097–1110 (2012)
9. Chang, X.: Network simulations with OPNET. In: WSC 1999. 1999 Winter Simulation Conference Proceedings. Simulation-A Bridge to the Future (Cat. No. 99CH37038), vol. 1, pp. 307–314. IEEE (1999)
10. Coskun, A.K., Rosing, T.S., Whisnant, K.: Temperature aware task scheduling in MPSoCs. In: 2007 Design, Automation & Test in Europe Conference & Exhibition. pp. 1–6. IEEE (2007)
11. Cui, W., Kim, Y., Rosing, T.S.: Cross-platform machine learning characterization for task allocation in IoT ecosystems. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), pp. 1–7. IEEE (2017)
12. Ergun, K., Ayoub, R., Mercati, P., Rosing, T.: Dynamic optimization of battery health in IoT networks. In: 2019 IEEE 37th International Conference on Computer Design (ICCD), pp. 648–655, November 2019. https://doi.org/10.1109/ICCD46524.2019.00093

13. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017)
14. International Data Corporation: The Growth in Connected IoT Devices (2019). https://www.idc.com/getdoc.jsp?containerId=prUS45213219
15. Chang, J.-H., Tassiulas, L.: Maximum lifetime routing in wireless sensor networks. IEEE/ACM Trans. Netw. **12**(4), 609–619 (2004). https://doi.org/10.1109/TNET.2004.833122
16. Karl, E., Blaauw, D., Sylvester, D., Mudge, T.: Reliability modeling and management in dynamic microprocessor-based systems. In: Proceedings of the 43rd annual Design Automation Conference, pp. 1057–1060. ACM (2006)
17. Mercati, P., Paterna, F., Bartolini, A., Benini, L., Rosing, T.Š.: WARM: workload-aware reliability management in Linux/Android. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **36**(9), 1557–1570 (2016)
18. Minakov, I., Passerone, R.: PASES: an energy-aware design space exploration framework for wireless sensor networks. J. Syst. Arch. **59**(8), 626–642 (2013)
19. Nikolaev, S., Banks, E., Barnes, P.D., Jefferson, D.R., Smith, S.: Pushing the envelope in distributed ns-3 simulations: one billion nodes. In: Proceedings of the 2015 Workshop on Ns-3, WNS3 2015, pp. 67–74. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2756509.2756525. https://doi.org/10.1145/2756509.2756525
20. Nikolaev, S., et al.: Performance of distributed ns-3 network simulator. In: Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, SimuTools 2013, pp. 17–23. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL (2013)
21. Samie, F., Tsoutsouras, V., Masouros, D., Bauer, L., Soudris, D., Henkel, J.: Fast operation mode selection for highly efficient IoT edge devices. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., 1 (2019). https://doi.org/10.1109/TCAD.2019.2897633
22. Sonmez, C., Ozgovde, A., Ersoy, C.: EdgeCloudSim: an environment for performance evaluation of edge computing systems. Trans. Emerg. Telecommun. Technol. **29**(11), e3493 (2018)
23. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A.: The case for lifetime reliability-aware microprocessors. In: ACM SIGARCH Computer Architecture News, vol. 32, p. 276. IEEE Computer Society (2004)
24. Stathis, J.H.: Physical and predictive models of ultrathin oxide reliability in CMOS devices and circuits. IEEE Trans. Device Mater. Reliab. **1**(1), 43–59 (2001)
25. Tapparello, C., Ayatollahi, H., Heinzelman, W.: Energy harvesting framework for network simulator 3 (ns-3). In: Proceedings of the 2nd International Workshop on Energy Neutral Sensing Systems, pp. 37–42. ACM (2014)
26. Thomas, A., Guo, Y., Kim, Y., Aksanli, B., Kumar, A., Rosing, T.S.: Hierarchical and distributed machine learning inference beyond the edge. In: 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), pp. 18–23, May 2019. https://doi.org/10.1109/ICNSC.2019.8743164
27. Wu, H., Nabar, S., Poovendran, R.: An energy framework for the network simulator 3 (ns-3). In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, pp. 222–230. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications) (2011)

28. Yao, S., Zhao, Y., Zhang, A., Su, L., Abdelzaher, T.: DeepIoT: compressing deep neural network structures for sensing systems with a compressor-critic framework. In: Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, pp. 1–14 (2017)
29. Zhai, D., Zhang, R., Cai, L., Li, B., Jiang, Y.: Energy-efficient user scheduling and power allocation for NOMA-based wireless networks with massive IoT devices. IEEE Internet Things J. **5**(3), 1857–1868 (2018)
30. Zhang, H., Xiao, Y., Bu, S., Niyato, D., Yu, F.R., Han, Z.: Computing resource allocation in three-tier IoT fog networks: a joint optimization approach combining Stackelberg game and matching. IEEE Internet Things J. **4**(5), 1204–1215 (2017)
31. Zhang, L., et al.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 105–114. ACM (2010)
32. Zhuo, C., Sylvester, D., Blaauw, D.: Process variation and temperature-aware reliability management. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 580–585. European Design and Automation Association (2010)