ECM: Improving IoT Throughput with Energy-Aware Connection Management

Fatemeh Ghasemi, Lukas Liedtke and Magnus Jahre
Department of Computer Science, Norwegian University of Science and Technology (NTNU)
(fatemeh.ghasemi@ntnu.no, lukas.liedtke@ntnu.no, magnus.jahre@ntnu.no)

***Abstract*—Designing Internet of Things (IoT) devices that solely rely on energy harvesting is the most promising approach towards achieving a scalable and sustainable IoT. The power output of energy harvesters can however vary significantly and maximizing throughput hence requires adapting application behavior to match the harvester's current power output. In this work, we focus on the connection policy of the IoT device and find that the on-demand connect policy — which is used by state-of-the-art IoT runtime systems — and the aggressive maintain connection policy both fall short across a broad range of harvester power outputs. We therefore propose Energy-aware Connection Management (ECM) which tunes the connection policy and sampling frequency to consistently achieve high throughput. ECM accomplishes this by predicting both the average power output of the harvester and the energy consumed by the IoT device with a lightweight analytical model that only requires tracking six energy thresholds. Our evaluation demonstrates that ECM can improve throughput substantially, i.e., by up to 9.5× and 3.0× compared to the on-demand connect and maintain connection policies, respectively.**

***Index Terms*—Energy harvesting, IoT, connection management**

I. Introduction

The number of Internet of Things (IoT) devices is expected to grow substantially in the near future. Relying on battery-powered IoT devices unfortunately limits scalability — because replacing batteries is tedious and expensive at scale — and it is not sustainable — because battery production and disposal has a significant environmental footprint. Energy-harvesting IoT devices address these challenges by harvesting energy from for instance light, vibration, temperature gradients, or wireless transmissions [10]. Energy and throughput are however fundamentally intertwined in energy-harvesting systems [5], and they hence need to strive to make the most out of the harvested energy. Device power consumption should ideally match the power output of the energy harvester, in which case the system is energy-neutral [8], but this is challenging in practice due to variability in energy supply and demand.

IoT applications typically consist of different tasks, for example capturing a sample from a sensor or communicating one or more samples to the back-end system. An important design objective is to maximize throughput under an (implicit) energy constraint, i.e., maximizing the number of bytes of sample information delivered to the back-end system per unit time given the amount of energy that the harvester supplies. We find that IoT system throughput critically depends on how the system treats *management tasks* such as connecting to the back-end system or transmitting packets to keep the connection alive. The key challenge is that management tasks must be executed for the system to work as intended, but they do not directly contribute to throughput. Adopting policies that require executing too many management tasks hence yields suboptimal throughput because management tasks consume energy that could have been used to collect or communicate sample data.

To maximize throughput, IoT systems should therefore adopt task scheduling policies that minimize the amount of energy spent on management tasks. State-of-the-art Capybara [3] and Morphy [16] use *energy-greedy task scheduling* in which each task is associated with a worst-case energy cost. The energy harvesting subsystem then notifies the application System-on-Chip (SoC) when the system has stored sufficient energy to fully execute each task, thereby avoiding shutdowns due to lack of energy. Energy-greedy task schedulers treat management tasks inefficiently because they effectively adopt an *On-demand Connect (OC)* strategy, i.e., they wait until the system has harvested sufficient energy to connect and transmit sample data and then disconnect. OC is the most efficient policy when energy is scarce — because it avoids wasting energy on keeping a doomed connection alive — but yields suboptimal throughput otherwise — because the system wastes energy on connecting and disconnecting even if the system has sufficient energy to maintain the connection. A straightforward alternative is to adopt a *Maintain Connection (MC)* policy that always attempts to keep the connection alive. MC can perform (slightly) worse than OC when energy is scarce — because it may waste energy on executing tasks that do not complete as well as on keeping doomed connections alive — but it provides higher throughput than OC otherwise — because it amortizes connection overhead across multiple samples.

Our key observation is that OC and MC treat management tasks suboptimally and therefore leave substantial throughput on the table. More specifically, we find that once connected, the IoT system should carefully weigh the potential for improving throughput against the overhead of maintaining the connection. We hence propose *Energy-aware Connection Management (ECM)* which uses a light-weight analytical model to decide if the connection should be maintained or not, and, if it decides to maintain the connection, ECM tunes the sampling period to match the predicted power consumption of the SoC to the predicted power output of the energy harvester. In this way, ECM configures the system to adopt a near-energy-

This work is funded in part by the Research Council of Norway (Grant #286596) and the European Union through Horizon 2020 (Grant #101034240).

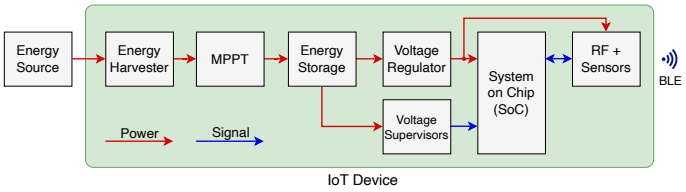


Fig. 1: Baseline IoT device. *The voltage supervisors enable implementing energy-greedy task scheduling in software.*

neutral operating point which in turn minimizes the number of management tasks, i.e., the number of times the IoT device must connect to the back-end system. Our evaluation covers a broad range of harvester configurations and demonstrates that ECM improves throughput by up to 8.0× (3.0×), 2.0× (1.4×), and 9.5× (3.0×) compared to OC (MC) for our benchmarks *ProtoNN*, *Bonsai*, and *Glucose*, respectively.

II. Background

Figure 1 provides an overview of our baseline IoT platform which is in line with prior work [3], [15]. We assume a state-of-the-art energy-greedy task scheduler inspired by Capybara [3] which runs on the SoC and dispatches the next task when the task's low-power voltage supervisor detects that the platform has stored sufficient energy to execute it successfully. The remaining platform components collaborate to efficiently power the SoC. The energy harvester converts ambient energy into electrical energy which is then fed to the Maximum Power-Point Tracker (MPPT) which optimizes harvester efficiency by adjusting its output voltage such that it achieves the highest possible output power under the current ambient energy conditions. The device includes a voltage regulator between the energy storage and the SoC. We model a 2.2 mF capacitor as energy storage and a Microchip MCP1640 voltage regulator. The MCP1640 improves energy utilization by boosting the voltage of the capacitor to match the supply voltage of the SoC in the (typical) situation where the capacitor voltage is (much) lower than the SoC's required supply voltage.

Greedy task schedulers require measuring the amount of stored energy. One option is to directly measure the instantaneous power output from the harvester, e.g., by using shunt resistors and Analog-to-Digital Converters (ADCs). Although very low-power ADCs and current-sense amplifiers are available (e.g., TI ADS7112 and Maxim MAX9634), their power overhead is at least in the range of several micro-watts (depending on the operating mode). Prior work [3], [16] instead use voltage supervisors to track the voltage of the energy storage capacitor(s) at (much) lower power overhead. We model a Renesas ISL8800x series voltage supervisor for which the typical current consumption is 200 nA when operating at up to 3.3 V. Our baseline energy-greedy scheduler requires four voltage supervisors to support all tasks.

III. Energy-Aware Connection Management (ECM)

We now present ECM which improves upon state-of-the-art energy-greedy schedulers by tuning the connection policy and the sampling period to the current harvester power output.

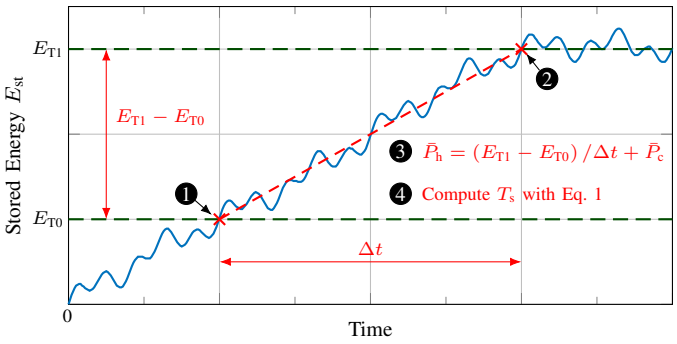


Fig. 2: ECM example. *ECM achieves near-energy-neutral operation by first predicting the average power output of the energy harvester $\bar{P}_h$ and then predicting a sampling period T_s that balances energy supply and demand.*

A. ECM Runtime Behavior

The core of ECM is its analytical energy model which predicts whether the connection should be maintained or not, and, if ECM detects that maintaining the connection is favorable, it selects a sampling period that achieves near-energy-neutral operation. Our fundamental insight is that this decision is governed by a simple equation:

$$T_s = \frac{E_c^*}{\bar{P}_h - \bar{P}_c^*}. \quad (1)$$

Equation 1 predicts the sampling period T_s at which the energy cost of work-proportional activities E_c^* is balanced against the predicted average power output of the harvester $\bar{P}_h$ minus the average power consumed by time-proportional activities $\bar{P}_c^*$. We will derive Equation 1 in Section III-B, but, before attending to this, we will describe how ECM uses Equation 1 at runtime. For now, it is sufficient to know that E_c^* and $\bar{P}_c^*$ are fully determined at design time and hence constants at runtime.

Example. Figure 2 exemplifies the main operation of ECM. For the purpose of the example, we assume that (i) the IoT device is initially not connected to the back-end system, and (ii) the upcoming task requires communication. We focus on the energy thresholds E_{T0} and E_{T1} which signify that the device has stored sufficient energy to execute task set T0 and task set T1, respectively. Different task types can share energy thresholds if their worst-case energy consumption is sufficiently similar.

The energy harvesting subsystem interrupts the SoC when an energy threshold is crossed. At ❶, the amount of stored energy crosses the threshold E_{T0}. Since the next task is a communication task, the device does not yet have sufficient energy to execute it. ECM however exploits that the amount of energy storage is known at this point in time and hence starts a timer. When the amount of stored energy later crosses the communication task threshold E_{T1} (see ❷), ECM inspects the timer and thereby measures the amount of time Δt that has passed since crossing E_{T0}. Since E_{T1} and E_{T0} are known, we know how much energy the system gained (or lost) across Δt.

ECM keeps a record of which tasks were executed during Δt and knows the worst-case energy consumption of each task type; this is required to perform energy-greedy scheduling. ECM can hence predict the average power consumption of the

SoC $\bar{P}_{\text{c}}$, and with $\bar{P}_{\text{c}}$ in place, we can combine it with the thresholds (E_{T0} and E_{T1}) and Δt to predict $\bar{P}_{\text{h}}$ (see ❸):

$$\bar{P}_{\text{h}} = (E_{\text{T1}} - E_{\text{T0}})/\Delta t + \bar{P}_{\text{c}}. \quad (2)$$

In other words, we exploit that the change in stored energy across Δt is determined by the relative difference between the average harvester power output $\bar{P}_{\text{h}}$ and the average power consumption of the SoC $\bar{P}_{\text{c}}$.

Since E^*_{c} and $\bar{P}^*_{\text{c}}$ are determined at design time, ECM has now captured the necessary information to compute T_s with Equation 1 ❹. If $\bar{P}^*_{\text{c}}$ is greater than $\bar{P}_{\text{h}}$, T_{s} is negative which indicates that the harvester's power output is insufficient to maintain the connection. In this case, ECM adopts an on-demand connect strategy, i.e., it connects to the back-end system, transfers the captured sample data, and then disconnects. Otherwise, T_s is positive, and ECM attempts to maintain the connection with a sampling period of at most T_s. ECM recomputes Equation 1 every time a threshold is crossed after connecting to the back-end system and updates T_{s} accordingly, thereby adapting to significant changes in harvester output power.

Optimizations. Equation 1 returns a large T_s when the predicted power output of the energy harvester $\bar{P}_{\text{h}}$ is only slightly larger than $\bar{P}^*_{\text{c}}$, i.e., the energy cost of keeping the connection alive is lower than the cost of reconnecting over a long time horizon. A large T_s can be inefficient because keeping the connection alive requires transmitting a keep-alive packet every 100 ms in our setup, and energy is then predominantly spent on keep-alive packets rather than sampling and transmission of sample data. ECM lets the developer tune this behavior by providing an upper bound on T_{s} called $T_{\text{s-max}}$. (We will explain how to derive $T_{\text{s-max}}$ at the end of Section III-B.)

When the energy storage is full, the system will lose some of the harvested energy. This constitutes a loss of throughput if the system could have used this energy to sample or communicate. ECM therefore includes an additional high-energy threshold above which it samples and communicates continuously. We set this threshold to 58% of the device's energy storage capacity because this is close to the mid-point between the communication threshold and maximum storage capacity. We also found it beneficial to include an additional energy threshold between the sampling threshold and the communication threshold to obtain a better estimate of harvester output power when connecting. ECM hence requires two voltage supervisors in addition to the four supervisors in the baseline, yielding an additional overhead of 400 nA (see Section II). We faithfully model this overhead in our evaluation.

B. Sampling Period Prediction

Having explained how ECM uses Equation 1 at runtime, we now turn our attention to explaining how we derived the equation and precisely defining E^*_{c} and $\bar{P}^*_{\text{c}}$. Previous work demonstrated that the energy consumption of IoT applications can be accurately predicted by capturing key characteristics of the application and IoT device [5], [12], and we build upon the insights of these works to predict T_{s}.

Overall energy consumption. Since ECM builds upon an energy-greedy scheduling baseline, we know that the device has sufficient energy to successfully connect to the back-end system and transmit the currently captured sample data when executing the connection task. The energy cost of these operations is hence sunk, and the task at hand is to determine a sampling period (if any) at which future consumed energy matches the predicted energy output of the harvester. We find that post-connect SoC energy consumption E_{c} can be modeled as the sum of the energy consumed by four key activities:

$$E_{\text{c}} = E_{\text{s}} + E_{\text{t}} + E_{\text{k}} + E_{\text{i}}. \quad (3)$$

More specifically, E_{c} is the sum of the energy consumed while collecting and processing samples E_{s}, transmitting sample data to the back-end system E_{t}, transmitting keep-alive packets to keep the connection alive E_{k}, and while idle in a low-power mode E_{i}. We model the energy consumption across a fixed time window t_{tot} which is the product of the sampling period T_{s} and the total number of samples $n_{\text{s-tot}}$ (i.e., $t_{\text{tot}} = T_{\text{s}} \times n_{\text{s-tot}}$).

Sampling and processing. We predict the energy consumption during sample collection and processing as the product of the average power consumption of the SoC while sampling $\bar{P}_{\text{s}}$, the time it takes to collect a sample t_{s}, and the number of samples in the time window $n_{\text{s-tot}}$:

$$E_{\text{s}} = n_{\text{s-tot}} \times \bar{P}_{\text{s}} \times t_{\text{s}} = n_{\text{s-tot}} \times E^*_{\text{s}}. \quad (4)$$

For the purpose of cleanly deriving Equation 1, we simplify Equation 4 by representing $\bar{P}_{\text{s}} \times t_{\text{s}}$ as E^*_{s}.

Data transfer. We focus on protocols where data transmission is organized in packets, and determining the total energy cost of data transmission hence requires computing the number of data packets. Each sample produces a certain amount of data d_{s} measured in bytes. Dividing d_{s} by the packet payload size d_{p} yields the number of packets required to transfer each sample. Applications may transfer multiple samples in a communication event, and prior work refers to this as the communication incidence n_{s} [5]. The total number of packets n_{p} in a communication event is hence $\lceil n_{\text{s}} \times (d_{\text{s}}/d_{\text{p}}) \rceil$, and we are ready to predict transfer energy E_{t}:

$$E_{\text{t}} = n_{\text{p}} \times \bar{P}_{\text{t}} \times t_{\text{t}} \times (n_{\text{s-tot}}/n_{\text{s}}) = n_{\text{s-tot}} \times E^*_{\text{t}}. \quad (5)$$

Equation 5 states that E_{t} is the product of the number of packets n_{p}, the average power consumption of the SoC during transmission $\bar{P}_{\text{t}}$, the time it takes to transmit a single packet t_{t}, and the number of communication events in the time window $(n_{\text{s-tot}}/n_{\text{s}})$. Again, we simplify the equation by combining all terms except $n_{\text{s-tot}}$ in E^*_{t}.

Keep-alive packets. Connection-oriented protocols typically require communication at specific intervals to keep the connection alive. In Bluetooth Low Energy (BLE), the minimum frequency f_{k} at which packets have to be sent is determined by the parameters connection interval and peripheral latency. If the application communicates at a lower frequency, it must transmit empty keep-alive packets. We model keep-alive energy E_{k} as the product of the average power consumption during

transmission $\bar{P}_{\text{k}}$, the transmission time of a single packet t_{k}, and the number of keep-alive packets n_{k} in the time window:

$$\begin{aligned} E_{\text{k}} &= n_{\text{k}} \times \bar{P}_{\text{k}} \times t_{\text{k}} \\ &= \left(t_{\text{tot}} \times f_{\text{k}} - \frac{n_{\text{s-tot}}}{n_{\text{s}}} \times \left\lceil \frac{n_{\text{p}}}{n_{\text{p-max}}} \right\rceil \right) \times \bar{P}_{\text{k}} \times t_{\text{k}} \\ &= n_{\text{s-tot}} \times E^*_{\text{k}} + t_{\text{tot}} \times \bar{P}^*_{\text{k}}. \end{aligned} \tag{6}$$

The number of keep-alive packets n_{k} is the maximum number of potential keep-alive transmissions in the time window (i.e., $t_{\text{tot}} \times f_{\text{k}}$) minus the number of communication events in which useful data is transmitted. The number of packets that can be transmitted between two potential keep-alive events is limited by $n_{\text{p-max}}$ which is a protocol parameter. Dividing the total number of packets n_{p} by $n_{\text{p-max}}$ hence takes care of the situation in which a single communication event replaces more than one keep-alive packet. To prepare for the upcoming derivation of Equation 1, we abstract the work-proportional energy cost and the time-proportional power cost of the keep-alive packets into the variables E^*_{k} and $\bar{P}^*_{\text{k}}$, respectively.

Idle. When the SoC is neither sampling nor transmitting data or keep-alive packets, it is idle in a low-power sleep mode. Idle energy consumption E_{i} is therefore total time t_{tot} minus active time multiplied by the average idle power consumption $\bar{P}_{\text{i}}$:

$$\begin{aligned} E_{\text{i}} &= \left[t_{\text{tot}} - \left(n_{\text{s-tot}} \times t_{\text{s}} + \frac{n_{\text{s-tot}}}{n_{\text{s}}} \times n_{\text{p}} \times t_{\text{t}} + n_{\text{k}} \times t_{\text{k}} \right) \right] \times \bar{P}_{\text{i}} \\ &= n_{\text{s-tot}} \times E^*_{\text{i}} + t_{\text{tot}} \times \bar{P}^*_{\text{i}}. \end{aligned} \tag{7}$$

We combine the work-proportional energy costs of being idle into the variable E^*_{i} and represent the time-proportional power costs of idle time by the variable $\bar{P}^*_{\text{i}}$.

Solving for the sampling period. We can now substitute the simplified forms of Equations 4, 5, 6, and 7 into Equation 3 and simplify by combining the work-proportional energy costs into E^*_{c} and time-proportional power costs into $\bar{P}^*_{\text{c}}$:

$$\begin{aligned} E_{\text{c}} &= n_{\text{s-tot}} \times (E^*_{\text{s}} + E^*_{\text{t}} + E^*_{\text{k}} + E^*_{\text{i}}) + t_{\text{tot}} \times (\bar{P}^*_{\text{k}} + \bar{P}^*_{\text{i}}) \\ &= n_{\text{s-tot}} \times E^*_{\text{c}} + t_{\text{tot}} \times \bar{P}^*_{\text{c}}. \end{aligned} \tag{8}$$

The system is energy-neutral when all harvested energy is consumed (i.e., $\bar{P}_{\text{h}} \times t_{\text{tot}} = E_{\text{c}}$), and, by definition, $n_{\text{s-tot}}$ equals $t_{\text{tot}}/T_{\text{s}}$. Substituting these relations into Equation 8 results in t_{tot} cancelling, and by reorganizing we arrive at Equation 1:

$$\begin{aligned} \bar{P}_{\text{h}} \times t_{\text{tot}} &= (t_{\text{tot}}/T_{\text{s}}) \times E^*_{\text{c}} + t_{\text{tot}} \times \bar{P}^*_{\text{c}} \\ T_{\text{s}} &= \frac{E^*_{\text{c}}}{\bar{P}_{\text{h}} - \bar{P}^*_{\text{c}}} \end{aligned} \tag{9}$$

Bounding keep-alive overhead. To bound the overhead of keep-alive packets, the developer computes the ratio r of non-idle energy used on sampling and sample transmission to total non-idle energy for integer numbers of keep-alive packets n_{k}:

$$r = \frac{E_s + E_t}{n_{\text{k}} \times \bar{P}_{\text{k}} \times t_{\text{k}} + E_s + E_t} \tag{10}$$

The ratio r is a number between zero and one where zero means that all energy is spent on keep-alive packets, whereas r is equal to one when n_{k} equals zero. The developer can hence set a limit on r that is appropriate for the application and deployment. The maximum sampling period $T_{\text{s-max}}$ is then n_{k} divided by the minimum frequency f_{k} at which packets must be sent to keep the connection alive (i.e., $T_{\text{s-max}} = n_{\text{k}}/f_{\text{k}}$). In this work, we require r to be above 0.7 for all benchmarks, yielding $T_{\text{s-max}}$ of 200 ms for *ProtoNN* and *Glucose* and 900 ms for *Bonsai*.

IV. Experimental Setup

Our application SoC is the Nordic Semiconductor nRF52832 which features a single ARM Cortex M4 processor, an integrated BLE radio subsystem, 512 kB of non-volatile Flash memory, and 64 kB of volatile SRAM memory. The system operates at a clock frequency of 64 MHz. We use Nordic Semiconductor's S132 BLE protocol stack, and our back-end system is an Android mobile phone running nRF Connect. We disable all unused SoC components to save power.

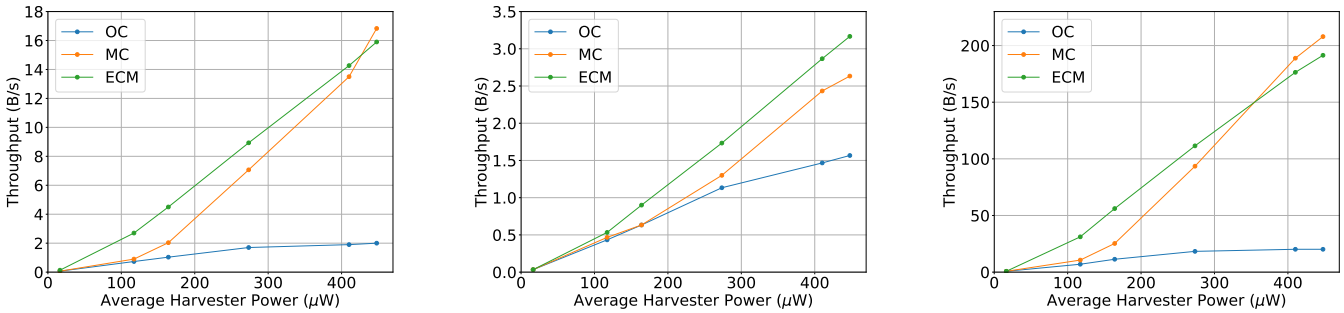
We model the energy harvesting subsystem within ESS [4] and configure ESS to take the outdoor solar energy traces provided by Kraemer et al. [9] as input. Outdoor solar energy harvesters follow a diurnal pattern because their power output is proportional to solar irradiance which peaks at mid-day (in the absence of weather-induced variation) and is zero between sunset and sunrise. We scale the power output of the energy traces to model a range of solar panels with varying sizes and efficiencies while retaining variability due to time of day and weather, ultimately yielding six harvester configurations with average power outputs of 16, 117, 164, 273, 410, and 448 µW; our trace covers three consecutive days in August 2018.

We compress the time scale of the energy trace by 600× to make evaluation tractable, i.e., 24 hours of energy trace data is represented by 2.4 minutes of evaluation time. This is not to the benefit of ECM because it is more sensitive to short-term variation in harvester power output than OC and MC. We evaluate all configurations for 10 minutes out of which 7.2 minutes represents 72 hours from the energy trace. We keep harvester output at zero during the final 2.8 minutes to ensure that all configurations consume as much as possible of the harvested energy. All experiments are initialized with zero stored energy.

We evaluate three IoT benchmarks. *ProtoNN* and *Bonsai* [6] are machine learning classifiers optimized for IoT devices, and *Glucose* is a health-monitoring application taken from the Nordic SDK. Our throughput metric is bytes per second (B/s), i.e., the number of bytes of sample data that the IoT device delivers to the back-end system per unit time.

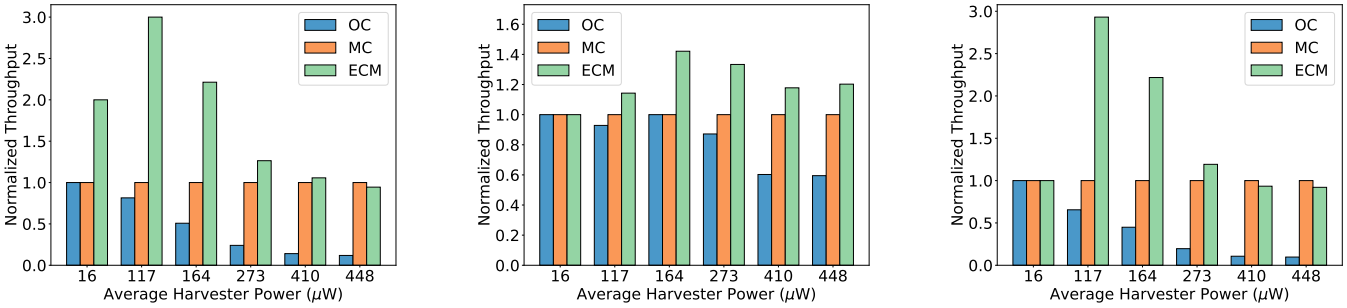
V. Results

Throughput. Figure 3 reports throughput with OC, MC, and ECM for *ProtoNN*, *Bonsai*, and *Glucose* across our solar panel configurations. Since throughput increases with energy, we augment Figure 3 with Figure 4 which reports throughput normalized to MC with each solar panel to make it easier to compare the connection managers to each other. The throughput difference between benchmarks is due to each benchmark requiring a different amount of energy to capture and transmit each sample. *Glucose* uses the least amount of energy and



(a) ProtoNN (b) Bonsai (c) Glucose

Fig. 3: Throughput versus energy. *ECM improves throughput compared to OC and MC across a broad range of harvesters.*



(a) ProtoNN (b) Bonsai (c) Glucose

Fig. 4: OC, MC, and ECM throughput normalized to MC. *ECM improves throughput substantially compared to OC and MC.*

hence achieves the highest throughput, whereas sampling is costly for *Bonsai* which in turn yields (much) lower throughput.

When the average harvester power output is low compared to the energy required to capture and transmit samples, OC, MC, and ECM yield similar throughput. This is the case for *Bonsai* with the 16 µW solar panel (see Figure 4b). The reason is that OC connects on demand and therefore does not waste energy on trying to maintain the connection, while ECM detects that the connection cannot be sustained and hence adopts the on-demand connect strategy. MC on the other hand tries to maintain the connection but quickly runs out of energy. We observe the same result with 16 µW *Glucose*, but in this case ECM was able to amortize the connection overhead over two samples whereas OC and MC were not. ECM was however not able to capitalize on this advantage within the evaluation time window, resulting in it completing the evaluation with more stored energy than OC and MC.

Figures 3 and 4 shows that OC and MC fall short for solar panels with moderate output power. OC yields low throughput because it is too conservative, i.e., it disconnects from the back-end system even if the system has sufficient energy to transmit more samples. MC on the other hand is too aggressive and causes the system to run out of energy because it samples and transmits too frequently to be sustainable. ECM achieves the best of both worlds by using its analytical model to select a favorable connection strategy and sampling period. Figure 4 demonstrates that ECM substantially improves throughput over OC and MC. For OC, ECM achieves its maximal throughput improvement with the 448 µW solar panel configuration, yielding speed-ups of 8.0×, 2.0×, and 9.5× for *ProtoNN*, *Bonsai*, and *Glucose*, respectively. Compared to MC, ECM improves throughput by 3.0×, 1.4×, and 3.0× for *ProtoNN*, *Bonsai*, and *Glucose*, respectively. ECM achieves its maximum speed-up over MC with the 117 µW panel for *ProtoNN* and *Glucose* and the 164 µW panel for *Bonsai*.

MC is the highest performer when energy is abundant, see *ProtoNN* with the 448 µW solar panel and *Glucose* with the 410 µW and 448 µW panels; this trend also holds for configurations with higher power output. MC outperforms ECM because we start our evaluation at midnight with zero stored energy, and, since outdoor solar energy follows a diurnal pattern, the power output of the harvester increases rapidly in the morning. This results in ECM initially underestimating harvester power output and selecting a higher-than-ideal sampling period — which in turn causes it to spend energy on keep-alive packets rather than sampling and data transfer — and explains why ECM yields lower throughput than MC in this case. Selecting (relatively) high-power harvesters do however incur overheads such as physical size and cost and are hence unattractive in many deployments. Moreover, the throughput difference between MC and ECM is small, e.g., *ProtoNN* with ECM provides only 5.5% lower throughput than MC with the 448 µW panel.

Energy consumption analysis. To demonstrate why ECM outperforms OC and MC, Figure 5 reports a break down of the SoC energy consumption for *ProtoNN* with the 117 µW solar panel. Since we supply the same amount of energy to all configurations, the total energy consumption under OC and MC is identical, whereas ECM's total energy consumption is 1.1% lower because this energy was consumed in ECM's two additional voltage supervisors and hence did not reach the SoC. The *Idle Sleep* category is energy spent in a low-power sleep mode in which interrupts and timers are enabled whereas

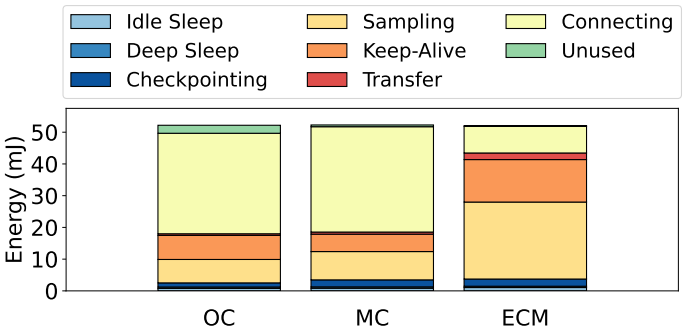


Fig. 5: *ProtoNN* energy breakdown with the 117 µW solar panel configuration. *ECM outperforms MC and OC because it minimizes the amount of energy spent on management tasks.*

only the wake-on-interrupt circuitry is enabled in the *Deep Sleep* mode. *Checkpointing* covers energy spent on backing up application state to non-volatile memory when the amount of stored energy reaches the backup energy threshold as well as restoring state once sufficient energy has been harvested. *Sampling* is energy spent on capturing samples and *Keep-Alive*, *Transfer*, and *Connecting* captures the energy spent on keep-alive packets, data packets, and connecting to the back-end system, respectively. Finally, *Unused* is energy that cannot be used because it is insufficient to execute the next task.

Figure 5 demonstrates that ECM outperforms OC and MC because it spends more of the available energy on collecting samples and transferring samples to the back-end system, i.e., the *Sampling* and *Transfer* categories are significantly larger for ECM than they are for OC and MC. Moreover, ECM successfully keeps the connection alive and hence spends much less energy in the *Connecting* category compared to OC and MC. This comes at the cost of ECM spending more energy in the *Keep-Alive* category, but this is a favorable trade-off because keep-alive packets overall require (much) less energy than reconnecting. OC interestingly has a non-negligible amount of *Unused* energy. This is the amount of energy left at the end of the evaluation where OC's next task is communication but it cannot be carried out because the amount of energy is insufficient to establish the connection. All configurations spend limited energy in the sleep modes and on checkpointing.

VI. Related Work

A large body of prior work focuses on adapting application behavior to the current energy conditions; Bakar et al. [1] provides an excellent overview. While early approaches simply execute when energy is available (e.g., Clank [7] and Hibernatus++ [2]), more recent approaches (e.g., CatNap [11], Camaroptera [14], and ePerceptive [13]) adapt application behavior to current energy conditions. REHASH [1] generalizes this approach by enabling developers to evaluate and deploy heuristic-based application adaptation strategies. These approaches are orthogonal to ECM, i.e., a developer can apply application-specific optimizations in conjunction with ECM's application-independent connection management optimizations.

VII. Conclusion

We have presented Energy-aware Connection Management (ECM) which adapts the connection policy and sampling period to achieve near-energy-neutral operation in energy-harvesting IoT systems. ECM uses an analytical energy model to predict a sampling period that balances the power consumption of the application SoC with the power output of the energy harvester or adopts an on-demand connect strategy if such a period cannot be found. Our evaluation demonstrates that ECM can improve throughput by up to 9.5× compared to the on-demand connect strategy used in state-of-the-art energy-greedy task schedulers.

References

[1] Abu Bakar, Alexander G Ross, Kasim Sinan Yildirim, and Josiah Hester. Rehash: A flexible, developer focused, heuristic adaptation platform for intermittently powered computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3), 2021.

[2] Domenico Balsamo, Alex S. Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M. Al-Hashimi, Geoff V. Merrett, and Luca Benini. Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12), 2016.

[3] Alexei Colin, Emily Ruppel, and Brandon Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[4] Fatemeh Ghasemi, Lukas Liedtke, and Magnus Jahre. ESS: Repeatable evaluation of energy harvesting subsystems for industry-grade IoT platforms. In *IEEE Int. Symp. on Workload Characterization (IISWC)*, 2023.

[5] Fatemeh Ghasemi, Lukas Liedtke, and Magnus Jahre. PES: An energy and throughput model for energy harvesting IoT systems. In *Int. Symp. on Performance Analysis of Systems and Software (ISPASS)*, 2023.

[6] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. Compiling KB-sized machine learning models to tiny IoT devices. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019.

[7] M. Hicks. Clank: Architectural support for intermittent computation. In *Proc. of the Int. Symp. on Computer Architecture (ISCA)*, 2017.

[8] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems*, 6(4), 2007.

[9] Frank Alexander Kraemer, David Palma, Anders Eivind Braten, and Doreid Ammar. Operationalizing solar energy predictions for sustainable, autonomous IoT device management. *IEEE IoT Journal*, 7(12), 2020.

[10] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[11] Kiwan Maeng and Brandon Lucia. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proc. of the Int. Conf. on Programming Language Design and Implementation (PLDI)*, 2020.

[12] J. San Miguel, K. Ganesan, M. Badr, C. Xia, R. Li, H. Hsiao, and N. Enright Jerger. The EH model: Early design space exploration of intermittent processor architectures. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[13] Alessandro Montanari, Manuja Sharma, Dainius Jenkus, Mohammed Alloulah, Lorena Qendro, and Fahim Kawsar. ePerceptive: Energy reactive embedded intelligence for batteryless sensors. In *Proceedings of the Conference on Embedded Networked Sensor Systems (SenSys)*, 2020.

[14] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems (ENSsys)*, 2019.

[15] Emily Ruppel, Milijana Surbatovich, Harsh Desai, Kiwan Maeng, and Brandon Lucia. An architectural charge management interface for energy-harvesting systems. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2022.

[16] Fan Yang, Ashok Samraj Thangarajan, Sam Michiels, Wouter Joosen, and Danny Hughes. Morphy: Software defined charge storage for the IoT. In *Proc. of the Conf. on Embedded Networked Sensor Sys. (SenSys)*, 2021.