A Schedule Method for Crawling Low-power IoT Devices

Li Meng*†, Cui Li*
*Institute of Computing Technology, Chinese Academy of Sciences
†Graduate University of the Chinese Academy of Sciences, China
{limeng, lcui}@ict.ac.cn

Abstract—**The search of Internet of things is an important way for people and applications to exploit useful sensors and device. In this paper, we study the crawling problem in IoT search engine, which is a fundamental component towards building a generic IoT search engine. With resource description method such as CoRE, it possible to build IoT search engines with Web technology, but the crawl of IoT sensors and device is very different from that of Web, especially when crawling target are energy-constraint and hibernate-enabled IoT sensors. These sensors are very common in IoT systems, and the efficiency of traditional crawling schedule strategy can be impact by their behaviors. In this paper, we formulate the IoT search problem as an optimization problem. We use the Poisson process to model events that user care, and proposed a measurement using the expectation of latency. Using this measurement as a main optimization index, an approach is designed to get the near optimal schedule strategy for different hibernate situations. At last, we use simulations to test the expectation of latency of our new scheduling method, which illustrates the effectiveness for repository freshness and lower energy impact on the IoT sensors.**

I. Introduction

Searching of IoT(Internet of Things) is a basic and important step for people and applications to find different kinds of sensors and IoT devices. A general propose IoT search engine can be used in many different scenes, which can improve the possibility of sharing sensors and device with others and significantly reduce the redundancy of sensor deploying, and different applications composition method (like IFTTT[**?**]) can be implied with the help of IoT search engine.

For example, in smart city applications, public sensors can be found in an ad hoc manner to provide predicates for traffic jam and queuing status. Smart home application need available sensors and devices to detect human activities and provide human-centric sensing, which calls for a realtime sensor discovery process. In recent future, unmanned systems like automatic driving systems and quad-copter delivery systems are very likely to be heavily dependent on information gathered from nearby sensors or other IoT devices, which also needs IoT search.

In these situations, an overall generic search engine is needed to collect the descriptions of these sensors, and send response to users' queries. Descriptions held by the sensors include not only static contents, but also dynamic contents which are generated by the sensor itself. We use the conception "description" as an abstraction of the state of a sensor. Indeed, many research group are working on dynamic describing methods, such as CoRE[1] from IETF and SensorML[2] from OGC.

We are building an IoT search system based on web search. We suppose every sensors and devices in this system supports IP protocol. Sensors and device can generate description file automatically, which may be events detected by these sensors, the most recent updates of data, or simply static meta information about the sensors and their functions. IoT crawlers crawl different sensors and devices for these descriptions files, and copy these into a local copy at server end, called repository. The repository is an organized storage, with whom the user queries about IoT sensors can be accomplished. The most obvious advantage of using this approach is that we don't need to invent wheels ourself. With maturity technology from information retrieval, there is no need to modify the server side architecture or protocols. There is even no need to modify the hardware of IoT sensors, if a sensor support protocols like CoAP.

But IoT search is different from the web search because the Web pages as the search target of Web search is more predictable than that of IoT search, and IoT search calls for higher real-time performance. In this paper we consider the possible energy constraints and hibernation behavior of sensors, which may correspondingly impact the battery life of sensors and result in invalid crawls. Information generated by sensors maybe out of date if the sensors are not crawled at a proper time. So, it's critical to choose crawling frequency and strategies for crawlers. In order to choose an appropriate strategy, we hold an assumption that the sensor message that user cares updated in a Poisson manner, and then design an update plan that maximum the expectation of the sum of the messages' freshness, which could be an important compromise between server storage and the aging of stream data generated by massive amount of sensors.

A. Related Work

Technologies as UPnP[3], Bonjour[4] and other prevalent network configure protocols are not suitable for upper applications, mainly because these technologies can only be used in a sub-net, and the description method of service in IoT devices are relatively weak. One dilemma of designing an IoT search system is choosing an appropriate crawling rate to update a certain device. Previous work[5][6][7] analyzed how the crawler strategy affect the freshness of the repository of web pages maintained by web crawlers, which determines the query hit rate and whether this system can return a up-to-date result. In a IoT search system, "up-to-date" requirements may change to an "up-to-minute" one. Moreover, some inherent properties of IoT like low power strategy and resource constraints, make

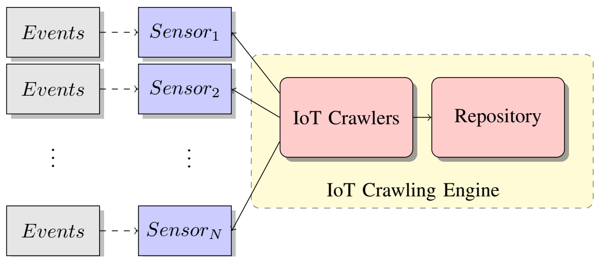

Fig. 1: Framework of IoT Search System

IoT crawler's strategy very different from the crawlers' in a traditional web search engine.

Due to the heterogeneous features and constraints of IoT devices, several different solutions have been proposed. Whether IoT search engine should be push-based or pull-based is a main dispution. In a push-based system, how to build a low latency search system is a prior. Technologies like TinyDB[8] and IoT feeds[9] takes IoT device as datasource in a database, and process structed command to query the information stream generated by IoT devices. But this method is hard to be implemented to a large network scale, as there are few consideration of heterogeneous of IoT sensors and devices: not all devices are capable to provide a continuous stream feed. This problem lay in an inner defect of the push-method approaches: the register behavior is unreliable and unpredictable. For a data consumer, they cannot feed any data source until data source are registered, but this register process can not be controlled by data consumer himself, this is obviously unreasonable. And it's also unnecessary for all the sensors and devices to generate data, which can be a heavy burden for both network and servers where these IoT devices are registered. This two problem affect the scalability and efficiency of a push-based IoT search system. Pull-based method can handle the unreliable register behavior. As most modern Web search engines, recent research prototype[10] of pull-based IoT search engine use crawlers and indexing method, along with semantic IoT technology as IoT device descriptions. Previous works of semantic IoT[11] and the propose of CoRE[1] devoted to standardize IoT device description languages, which indicated the possibility of building a web-style IoT search system. In this way, scalability of the system is extended, and more strategy can be implemented on this system to improve the efficiency, for example coordinators, say a smart gateway, can be add to provided dynamic data aggregations for sensors in a sub-net, which can be useful in an ultra-low power situations. At last, taking consideration of the cost of the integration of a IoT search system to Web, a web-style IoT search engine based on pull method is a rational choice.

B. *Framework*

We set our IoT search system with pull-based method. With the help of semantic IoT method and the practical protocols, IoT sensor messages can be converted to descriptions automatically at realtime. IoT search engine regularly updates the repository, which can be considered as a copy of descriptions of different sensors.

When building a prototype, we can use CoAP with 6LowPAN as the communication protocol with sensors and other IoT devices, the description message are list in the ./well-know file used in the CoRE framework, and this description can be update by the sensors in realtime. At what rate should the crawler crawl these useful messages in order to keep the freshness of repository? Although many previous work of Web search has been done, but IoT devices search with constraint power and duty cycle make IoT search quite a different work. In this paper, we will discuss the repository freshness problem, particularly, how to handle the sensor constraints and duty cycle stratergy widely used in IoT systems.

C. *Contribution*

This paper has two major contributions.

- This is the first paper to consider the crawling problem in IoT system, where we model the hibernation and energy constraint of sensors and model also model crawl scheduling as an optimization problem.
- A practical heuristic approach for choosing crawling strategy under different circumstances is proposed for this model, which is evaluated in simulation.

II. Formulation

Assume information updating events arrives at each sensor as a Poisson Process $\{N(t), t \geq 0\}$ having rate λ. These events can be written as random variable $U_1, U_2, \ldots$. The crawler access this sensor at time t. Each event happens before crawling time t and after last crawling time, which called t_s, cause the expectation of repository refresh latency to increase, which means the data gathered by sensor become old/out-of-fashion. Refresh latency is an important indicator for the synchronization level between repositories and sensors in IoT search engine.

We suppose there are totally N sensors and M time of crawls during the whole period $[0, T]$. Our goal is to find a strategy to schedule crawlers, that minimize the expectation of latency when an IoT device is crawled. Minimize the expectation of latency is indeed minimizing the out time level of data that collected by IoT device. Suppose the crawlers commit n crawling action during time period T. Under strategy ϕ, for each time the repository latency is $E(\phi)$, where $i = 1, \ldots, n$. ϕ is an access sequence for different senors, where each element $\phi_{i,j} \in 0, 1$. If sensor i is planned to be crawled at time j, then $\phi_{i,j} = 1$, where time j belongs to a discrete time table $\mathbf{T}$, which is all the possible time that strategy ϕ can be implied at. The access count of all the sensors during $[0, T]$ can be defined as $\|\phi\mathbf{T}\|_1$. We can define the count of sensor S_k's access time $\sum_{i=1}^{N} \phi_{i,k}$, just in the same way as the total time. Considering the difference between sensors, we define ω_i as the expectation weight of sensor i. Then the final object function can be evaluated as $\|\omega_i E(\phi)\|_1$.

In this problem, object function must follow several constraints. Server load and IoT devices' energy consumption are two main limitations to consider as discussed above. The energy constraint of the sensor is formalized directly: any IoT sensors cannot be accessed too frequently, which means any period T, the access time C_i for sensor i is limited to be accessed less than γ_i times. For the server load, our settings are similar to the formal research[6], where the crawlers from the server can not commit more that θ crawls during time period T. With the object function and constraints, we get the following optimization problem:

$$\begin{array}{ll} \min_{\phi} & \|\omega_i E(\phi)\|_1 \\ \text{s.t.} & \|\phi\|_1 \leq \theta \\ & \sum_{\forall t \in \mathbf{T}} \phi_{i,t} \leq \gamma_i \\ & i = 1, \ldots, N \end{array} \tag{1}$$

Problem (1) is a generalized formulation, which has not model information collecting action and hibernation behavior of IoT devices. Next, we will start with the simplest none-hibernate data collection model and look into the complete form of (1).

III. Optimal Strategy for Crawling Sensors

In last section, we build a simple model to minimize the expectation. Now we are going to design an optimal strategy for non-hibernate sensors and a near optimal crawling strategy for hibernate-enabled sensors. Then we'll look for a performance bound. We will also show that our method can be generalized to a none periodic hibernate situation, which can cover most situation of real world IoT implementation.

A. Non-hibernate sensors

In the situation when the sensors don't hibernate, an intuition is to evenly distributed the crawling action across the crawling schedule. Here we give a proof of this proposition.

Lemma 1: In a continuous situation where no sensor hibernates, the optimal strategy is to scatter the crawl action over the timeline $t \in [0, T]$ as evenly as possible, where T is a constant.

Proof: The evenly crawl proposition here can be proved by the convexity of $E_h[G(t)]$ [12]. Here we donate $E_h[G(t)]$ as $f(t)$ for convenience. We start the proof with a simple situation where only two crawling are allowed. As a result, the whole expectation gain during $[0, T]$, which is marked as $F(0, T)$, is divided into two part, $f(t)$ and $f(T-t)$. With the convexity of $f(t)$, we get $2f(T/2) < f(T/2 - \epsilon) + f(T/2 + \epsilon)$ for $\forall \epsilon > 0$. By substituting $F(T) = f(t) + f(T-t)$ into it, we get $F(T/2) < F(T/2 + \epsilon)$ for $\forall \epsilon > 0$, indicating $T/2$ is the minimum point of $F(t)$.

This conclusion can be generalized to a normal situation where multiply crawling are committed with the following induction. Now suppose there are n crawling action, among which first $n-1$ action evenly divide the timeline. The time length of n-th crawling action does not equal to the adjacent one's, and we claimed this strategy is optimal. For any adjacent pair divided by a single crawling action, if the division is not evenly, we can choose the evenly one as a new strategy, which can lead to a larger $F(t)$ according to the proposition above, which is a contradiction. ∎

With 1, we can get the exact increment of expectation when different crawling times are implied to a same sensor. This can be used to minimize the overall expectation of repository latency, which formalized as $\|\phi \mathbf{T}\|_1$ above. Here the relationship between crawling time and $E[\phi(n)]$ can be computed simply as follows, where $\phi(n)$ stands for the optimal strategy with n crawls, and T is the time period within consideration.

$$E[\phi(n)] = nE[G(\frac{T}{n})] \tag{2}$$

Function (2) can be rewrite to an close-form expression as $T(T\lambda/n - 1 + e^{-T\lambda/n})$, which is convex. with this function, the minimum expectation can be calculated directly.

B. Periodic Hibernate Sensors

Here we also study the relationship between n and the minimum latency expectation, but try to be more pragmatic. With the optimal strategy $\phi_i(n)$ for sensor i which is only related to the crawling time n, we can substitute functions (11) and (10) into (1) to get the latency expectation of non-hibernation version. But with hibernation, at some time points the sensors can not be accessed, resulting in a complex piecewise expectation function, which makes the relationship between n and $\phi_i(n)$ unclear and completely theoretical approach likely impossible. Here we focus on the situation when $T_w < T_h$ and $n < T/\Delta T$. Applications which hold $T_w \leq T_h$ is rare, while if $n < T/\Delta T$ is just the same as the push situation we talked before. A concrete formulation extends from 1 is as follows, where $\mathbf{t}$ is the set of target time sequence that a crawl action can be scheduled to, and $k \leq 0$.

$$\begin{array}{ll} \min_{\mathbf{t}} & E_f(T, t_n) + \sum_{i=1}^{n-1} E_f(t_{i+1}, t_i) \\ \text{s.t.} & t_1, t_2, \ldots, t_n \in \mathbf{t}, n > 0 \\ & 0 < t_1 < t_2 < \ldots < t_n < T \\ & t_i \in [t_s + k\Delta T, t_s + T_w + k\Delta T],\ t_i \in \mathbf{t} \end{array} \tag{3}$$

We first look into sub-problems in privilege to design algorithm for problem 3. Here an intuition of the problem is to arrange the crawls at the end of a work cycle, we show this intuition is indeed an optimal strategy when the crawl interval is an integral multiple of duty cycle ΔT. We uses the same symbols as the previous.

Lemma 2: If the time interval of an adjacent pair of crawls at t and $t + \Delta t$ is an integral multiple of duty cycle, then the optimal crawl time t follows $t \bmod \Delta T = T_w$.

Proof: The difference between expectations with two different time intervals, which have the different offset and have the same length multiple of duty cycle, can be written as $E_f(t, t_s) - E_f(t + \Delta t, t_s + \Delta t)$. The expansion of $E_f(t, t_s)$ is $E_h[G(t'')] - E[G(t)''] + E[G(t - t_s)]$, notice that $E_h[G(t'')]$ has a lower change rate compared with $E[G(t)]$, whose coefficient is $T_w/\Delta T$ instead of 1. This indicates that the previous difference is a non-increasing function, which the optimal is the minimum when $\Delta t = T_w$. ∎

The solution of the general situation is very similar to the previous when both t and $t + \Delta t$ locate in working cycles. However, when crawl $t + \Delta t$ locate in a hibernate cycle the expectation will become 0. Also, the result will become opposite when t locate in a hibernate cycle while $t + \Delta t$ locate in a working one. In this situation, the expectation will become larger with more time of work cycle included into the expectation area. With these observations, we can design a strategy that in some circumstances, the optimal solution can be retrieved.

Algorithm 1 Heuristic Method of Latency Minimum Periodic Crawl

```
Input: $n$, $T$, $T_w$, $T_h$
Output: $t_1, \ldots, t_n$
1: $m \leftarrow T/(\Delta T n)$, $r \leftarrow (T/\Delta T) \bmod n$
2: for $i = 1$ to $m$ do
3:     $t_{m-i+1} \leftarrow T - \Delta T i + T_w$
4:     if $i \neq 0 \wedge r \neq 0$ then
5:         $t_{m-i-1} \leftarrow t_{m-i+1} - \Delta T$
6:         $r \leftarrow r - 1$
7:     end if
8: end for
9: return $t_1, \ldots, t_n$
```

The result can be computed in one pass, so the complexity of this approach is $O(n)$, where n is the size of crawl times.

C. Non-periodic Hibernate Sensors

Our hibernation model is periodic and predictable in the previous example. However, in a real world IoT system, duty cycle in a hibernation mechanism is very likely to be non-periodic, and hibernation time will be the major action of the sensor, where $T_h \gg T_w$ holds. Also the crawling interval maybe much larger than T_h. If the length of working cycles are neglected and represented as time points, then problem of searching optimal crawling can be transformed to a search problem, which can be solved by dynamic programming.

Here a directed acyclic graph(DAG) model $G(E, V)$ is used to formalize the non-periodic hibernate problem. E are the edges of the graph, and V are the vertex. We suppose there are m vertex. If there are T' time points in the timeline which stand for the non-periodic working cycle. Corresponding, there are exactly T' vertex in this graph, from number 1 to number T. We define n as the count of crawls used, which indicates there are $n-1$ vertex allowed to pass, because the last vertex must be chosen. Now we take every points which stand for the working cycle as graph vertex, written as v_i. We defined that, for each pair of time points t_i and t_j, if $t_i < t_j$, a directed edge $e_{i,j}$ from v_i to v_j is added to the graph, where t_i and t_j are all . A weight value w_{ij} is set to $e_{i,j}$, whose value is $E_f(t_j, t_i)$, the expectation gain of a crawl at time t_j whose previous crawl is at t_i. Our problem is, how to find a path $e_{1,i}, \ldots, e_{j,T}$ from v_1 to v_m, which take exactly n steps, that minimum the sum of all the weights along. This transform process can be illustrated by Fig. 2.

With an acceptable scale of time range T and crawl time n, this problem can be solved by dynamic programming. The solution state with a vertex v_i and the available crawl time n' left can be used as an sub-problem. Noticing the optimal expectation can be computed by adding weight $w_{i,j}$ with all the expectation known in sub-problems. The recursive relationship is summarized as follows:

$$f(v_i, n') = \begin{cases} \min_{k=1}^{i-1}\{f(v_{i-k}, n'-1) + w_{i,i-k}\} \\ \text{if } i \in [1, m-1] \\ 0 \text{ if } n' = 0 \text{ and } i = 1 \\ \infty \text{ otherwise} \end{cases} \tag{4}$$

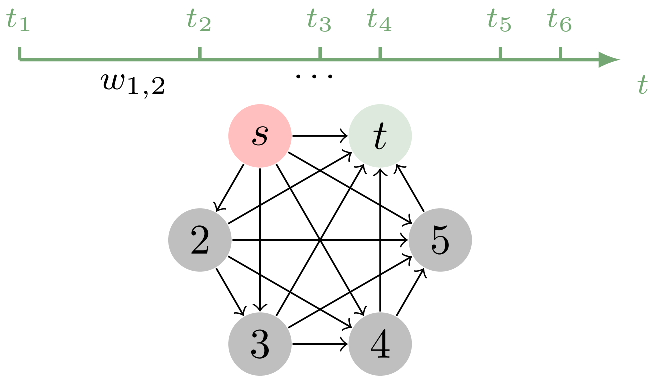

Fig. 2: Example of Non-periodic Hibernate Problem Transformation

$f(v_i, n')$ is the sum of weight of a path that start from v_1 and end with v_i, with cost exactly n steps. In each step, we go over the The computation complexity of iterating process is $O(T^2 n)$, in which nT sub-problem are computed and each computation will cost n operations. The backtrack method uses $p(v_i, n)$ to get the optimal strategy, which takes at most n steps. As a result, the overall complexity is still $O(T^2 n)$

Algorithm 2 Latency Minimum Non-periodic Crawl Method

```
Input: $G(V, E)$, $n$
Output: $t_1, \ldots, t_n$
1: $f(v_i, n') \leftarrow \infty$, $p(v_i, n') \leftarrow 0$
2: for $n' = 1$ to $n - 1$ do
3:     for $i = 2$ to $m$ do
4:         $f(v_i, n') \leftarrow \min_{k=1}^{i-1}\{f(v_{i-k}, n'-1) + w_{i,i-k}\}$
5:         $p(v_i, n') \leftarrow$ minimum $v_{i-k}$
6:     end for
7: end for
8: $t_n \leftarrow T$, $t_i \leftarrow$back trace from $p(v_i, n)$ for $t_i$
9: return $t_1, \ldots, t_n$
```

D. Global Solutions

In previous section, we figure out the relationship between the number of crawls and the expectation of latency $E_L(n)$. Although we did not find a close-form solution, our approximation method can meet the demand with an acceptable computation cost. The global optimization problem to dispatch crawls to different resources can be solved by dozen of optimization methods. Now the object function of formulation 1 can be rewrite as $\sum_{i=1}^{N} \omega_i E_L(n_i)$. The bandwidth constraint can be taken as $\sum_{i=1}^{N} n_i < M$ and the energy constraint of single sensor can be rewrite as $n_i < \gamma_i$.

Once crawl times n is chosen, then the optimal crawl strategy for a single sensor is determined, and every function of $E_L(n)$ is non-decreasing, we design a naive approach which is very similar to SMO[13]. The big figure is to dispatch crawl time and continuously refine the dispatch strategy until no improvement can be made. First we dispatch the M crawl time to N sensors according to ω_i. Then we start to search sensors for a pair i, j. If $i \leftarrow i-1$ and $j \leftarrow j+1$ will cause a largest improvement to the object function compared to the other pairs, then changes of i and j are conformed. Continue this process until none improvement can be made.

E. Parameter Selection

Through algorithm

IV. Simulation and Evaluation

A. Measurement

B. Parameters

C. Compare with Random Scheduling

V. Conclusion

Acknowledgment

The authors would like to thank...

References

[1] CoRE Working Group. Constrained RESTful Environments (CoRE) Link Format. *IETF CoRE*, pages 1–22, 2012.

[2] Mike Botts and Alexandre Robin. Opengis sensor model language (sensorml) implementation specification. *OpenGIS Implementation Specification OGC*, pages 07–000, 2007.

[3] UPnP Forum. http://www.upnp.org/. [Online].

[4] Apple Bonjour. https://www.apple.com/hk/en/support/bonjour/, 2002. [Online].

[5] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to improve freshness. *SIGMOD*, 29:117–128, 2000.

[6] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. *WWW*, page 136, 2002.

[7] James R. Challenger, Paul Dantzig, Arun Iyengar, Mark S. Squillante, and Li Zhang. Efficiently serving dynamic data at highly accessed web sites. *IEEE/ACM Transactions on Networking*, 12(2):233–246, 2004.

[8] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[9] Kamin Whitehouse, Feng Zhao, and Jie Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. In *EWSN*, pages 5–20, 2006.

[10] BM Elahi, Kay Römer, and B Ostermaier. Sensor ranking: A primitive for efficient content-based sensor search. *IPSN*, pages 217–228, 2009.

[11] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, 17:25–32, 2012.

[12] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[13] J Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. pages 1–21, 1998.

Appendix A
Analysis of Latency Expectation

A. Expectation of a Single Crawl

Our goal is to analyze the function property of the updates' freshness expectation at a certain time t. We define $E_f(t, t_s)$ as the expectation of latency when crawlers crawl IoT sensor at time t, and t_s is the last time this sensor crawled. When $t_s = 0$, we use $E[G(t)]$ for convenience.

We first model a very simple example when one crawler crawls an IoT sensor which act just as Web pages: have no access limits and no hibernation. The sensor start working at time 0, every events captured by the sensor is recognized as an update of sensor description, which the crawler can access at any time. The total freshness latency when crawl at time t can be considered as an random variable $G(t)$, so our goal can be rewritten to compute the close form of the expectation of $G(t)$, which can be written as $E(G(t))$. Notice $E(G(t))$ can be divide into different parts due to the count of events U_i happen during $[0, t)$, then $E(G(t))$ can be written as:

$$E[G(t)] = \sum_{n=1}^{\infty} E[G(t)|N(t) = n]P(N(t) = n) \tag{5}$$

where $P(N(t) = n)$ is the probability of having n events during $[0, t)$, which follows Poisson Distribution, holding $P(N(t) = n) = e^{-\lambda t}(\lambda t)^n/n!$. $E[G(t)|N(t) = n]$ can be resolved as:

$$\begin{aligned} E[G(t)|N(t) = n] &= E[\sum_{m=1}^{n} \lambda(t - t_m)e^{-\lambda U_m}] \\ &= \lambda \sum_{m=1}^{n} \int_0^t (t - t_m)e^{-\lambda t_m}\, dt_m \\ &= \frac{n}{\lambda}(t\lambda + e^{-\lambda t_m} - 1) \end{aligned} \tag{6}$$

t_m stands for the happened time of the m-th event. By taking (5) and (6) together, we get the close form solution of the expectation for the repository latency.

$$E[G(t)] = (e^{-t\lambda} - 1 + t\lambda)t \tag{7}$$

We can verify that equation (5) is convex at $[0, \infty)$ by computing its second derivative, which indicates that if crawlers fail to access the IoT sensors in a long period, then repository aging process would accelerate. Previous work[5] came to a similar result.

Next, we consider the hibernate situation, in which sensor's work schedule is a known preliminary. Sensor starts to work at time t_s, and begin to work at a fix duty cycle ΔT. $\Delta T = T_w + T_h$, where T_w is called a working cycle and T_w is called a hibernate cycle. Sensor can not collect any message in a hibernate cycle, and can not be accessed by a crawler. It's hard to directly deduce an expectation for a Poisson process through a scattered timeline. However, since we've got the close-form solution of the naive situation, we find that the expectation of hibernate situation can be pile up with different naive ones. Take the following toy example for clarity. We'd like to evaluate the expectation $E_h(G(t))$ of latency at time $t \in [t_s + \Delta T, t_s + \Delta T + T_w)$, which is the second work cycle of the sensor, then the following equation hold:

$$E_h[G(t)] = E[G(t - \Delta T)] - (E[G(t)] - E[G(t - T_w)])$$

Notice expectation $E_f[t, t_s]$ which takes start time t_s as parameter can be simply represent by $E_h[G(t - t_s)]$, so here we only discuss the evaluation of $E_h[G(t)]$, which is short for $E_h[G(t)|t_s = 0]$. Suppose start time $t_s = 0$, with the tricky method above, a general form of $E_h[G(t)|t_s = 0]$ can be

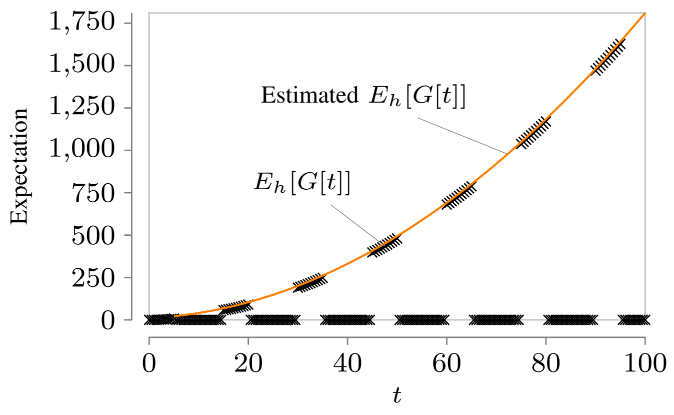

Fig. 3: Expectation of Repository Latency

induced as the following for any $t \in [t_s+k\Delta T, t_s+k\Delta T+T_w)$ where $k \in \mathbb{N}$.

$$\begin{aligned}E_h[G(t)] = E[G(t-K\Delta T)]+ \\ \sum_{i=1}^{K}(E[G(t-(K-i)\Delta T)]- \\ E[G(t-(K-i)\Delta T - T_w)])\end{aligned} \quad (8)$$

Here $K = \lfloor t/\Delta T \rfloor$, which is the count of integrated cycle ΔT. Notice that for any other t, $E_h[G(t)] = 0$, because crawlers are not allowed to access a hibernating sensor. For any time t in a sensor working cycle, by summing those expectations caused by working cycle before t, we can get the overall expectation of out-of-time level when considering sensor's hibernation. We create a sketch to illustrate the analyze result, where we use (8) to plot the expectation when a crawler crawls the IoT sensor at time t. We set the constraints $\lambda = 1/2$, which is the parameter for Poisson process, then set $t_s = 0$, $T_w = 5$ and $T_h = 10$.

With Fig.3, it's reasonable to guess that expectation at a working cycle is exponential, or at least near-exponential. In fact the none zero part of this curve is very similar to $E[G(t)]$. If this guess can be proven, then it's very likely that expression (8) and (11) can be simplified. Notice that the $E_h[G(t)]$ is a linear composition of different $E[G(t)]$, we verified incremental between two adjacent working cycle by computing $E[G(t)] - E[G(t-\Delta T)]$, and find these tedious expressions (8) can actually be represented by some simple ones. By imply (8) into this, we can get the following expression:

$$E_h[G(t)] - E_h[G(t-\Delta T)] = E[G(t)] - E[G(t-T_w)] \quad (9)$$

Expression (9) shows that the difference of an adjacent work cycle of $E_h[G(t)]$ has the same form of $E[G(t)]$. If an approximation are allowed, by taking the incremental of $E[G(t)]$ as the incremental of $E_h[G(t)]$, then equation (8) can be rewrite as follows:

$$E_s[G(t-T_w)] = \frac{T_w t}{\Delta T}(e^{-t\lambda} - 1 + t\lambda) \quad (10)$$

We use the token $E_s[G(t)]$ as the simplified version of (8). Notice that the domain of function (10) is $[t_s + k\Delta T, t_s + k\Delta T + T_w]$, where $k > 1$. We use the same λ as the previous example, and plot the estimated $E_h[G(t)]$ in Fig.3, which can make a sensible approximation. In fact, error bound of this approximation is quit straight forward, which is right beneath the value of $E_h[G(T_w)] - E_s[G(T)]$. If λ is large enough, the approximation will cause a large error when used to estimate the expectation of a working cycle. We use this method for final implementation, which is more computational friendly. The intuition of equation (9) and (10) is useful to interfere the optimal strategy for any hibernation sensors, even when the duty cycle is not periodic.

B. *Expectation of Crawls in a Time Period*

At last, we revisit the original problem of computation of $E_f(t, t_s)$. Although this problem can be represented by the above result, it is still trifle indeed. This problem be divided into two situation according to the value of t_s, if t_s is at a working cycle, the expectation can be estimate by $E_h[G(t)]$ which start at 0 and subtract the expectation latency cause by the events arriving at time $[0, t_s)$, which is $E_f(t, t_s) = E_h[G(t'')] - (E[G(t'')] - E[G(t-t_s)])]$.

We define $t' = t_s - t_s \bmod \Delta T$ and $t'' = t - \lfloor t_s/\Delta T \rfloor \Delta T$ for convenience.

Here the minuend means the expectation at time point t of latency during time period $[0, t_s]$. If t_s is at a hibernate cycle, then the result expectation is just as the one which t_s equals next duty cycle, which is $E_h[G(t')]$, which can be written as follows.

$$E_f(t, t_s) = \begin{cases} E_h[G(t')] \text{ if } t_s \bmod \Delta T \in [T_w, \Delta T) \\ E_h[G(t'')] - E[G(t'')] + E[G(t-t_s)] \\ \text{otherwise} \end{cases} \quad (11)$$

Combine expression (8) and (11), the expectation of freshness latency is defined in the problem of hibernate involved crawling.