Dribble: A learn-based timer scheme selector for mobility management in IoT

Bruno P. Santos[†,‡], Paulo H. Rettore[†], Luiz F. M. Vieira[†], and Antonio A. F. Loureiro[†]
[†]Department of Computer Science, Federal University of Minas Gerais, Brazil
[‡]Department of Computer and Systems, Federal University of Ouro Preto, Brazil
Email: {bruno.ps, rettore, lfvieira, loureiro}@dcc.ufmg.br

***Abstract*—In this work, we present Dribble a learn-based timer scheme selector to manage topology changes caused by mobility in the Internet of Things (IoT) context. IoT has turned smart devices part of our everyday lives. They are in everywhere with many shapes, sizes, and capabilities. For IoT to become even more ubiquitous, it is necessary to overcome the challenges posed by mobility. One of them is the management of topology changes, especially at the network layer. Currently, routing protocols check the topology through an advertisement timer scheme. Such schemes face a basic trade-off between being fast to find topology problems and concurrently be energy and overhead control saver. Although there are timer schemes designed to mobile context, all devices are governed by the same one, which is a hard assumption since IoT is heterogeneous and naturally, devices have different behaviors. Thus, Dribble learns the devices' mobility pattern and then it assigns a custom-made timer scheme conveniently for each device. Our results show that personalized timer schemes present better performance than single traditional timer schemes such as Trickle Timer (TT) and Reverse Trickle Timer (RevTT).**

***Index Terms*—IoT, Mobility, Trickle, Handoff management**

I. Introduction

Smart devices have begun to be part of our daily routine. They can be attached to infrastructures, wearable, and be moving by itself. When those devices are networked and connected to the Internet, they form the so-called Internet of Things (IoT). Nonetheless, they introduce new challenges from the network lens, because they are heterogeneous (e.g., TVs, smartphones, vehicles, etc.) and have different degrees of freedom concerning mobility. Thus, manageability and scalability are examples of key issues that ask for solutions, especially when the mobility factor is present. Until recently, most of IoT's proposed solutions were for static networks [1]. Only a few attempts took the mobile context into consideration [2], [3]. In the mobile and wireless environment, the routing protocol is a key component to enable mobility to the IoT. Mostly of routing protocols for mobile IoT have one timer scheme that governs the communication structure construction and maintenance by triggering from time to time control advertisements.

The timer scheme must deal fairly with a basic trade-off. If it is too greedy by sending advertisement frequently, it responds quickly to topology changes, but it spends energy and introduces an overhead to the wireless shared channel. However, if the timer scheme is too slow, it saves energy and bandwidth, but topology problems persist for a long time.

In order to balance this trade-off, we propose Dribble, a learn-based timer scheme selector. To the best of our knowledge, IoT networks are governed by a single timer scheme, without concerning devices mobility properties. Dribble differs from single timer schemes by setting a custom-made timer scheme to devices conveniently. It does that through a machine learning process that learns devices' mobility patterns and matches them to proper timer schemes. We evaluate Dribble against traditional timers such as Trickle Timer (TT) and Reverse Trickle Timer (RevTT) through extensive simulations. Our analyzes show that Dribble presents a better trade-off balance than a single timer scheme.

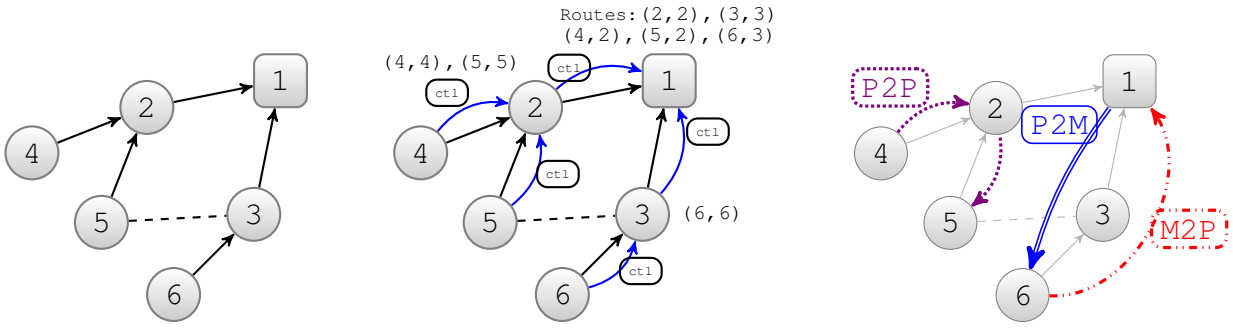
The rest of the paper is organized as follows. In Sec. II we review routing in IoT. Sec. III states the timer scheme problem and describes the related work. In Sec. IV, we present Dribble workflow. Dribble evaluation is detailed in Sec. V. Finally, in Sec. VI we highlight the final remarks and conclusions.

II. IoT Routing in a Nutshell

In IoT, mobility-enabled routing protocols build structures to perform multi-hop data forwarding in three main traffic patterns: Multipoint-to-Point (M2P), Point-to-Multipoint (P2M), and Point-to-Point (P2P). M2P is done by creating a tree or Directed Acyclic Graph (DAG) oriented towards a special device node (known as border router). By default, each node maintains at least one preferred parent, which is used to forward data (Fig. 1(a)). The M2P traffic pattern is cost-efficient in terms of memory requiring few routing states.

P2M is the dual traffic pattern (from root to nodes). Protocols that support P2M build paths by using extra control packets and routing states. Fig. 1(b) shows how a protocol can create P2M paths[1]. By supporting both previous traffic patterns, a protocol can combine them to provide P2P communication (Fig. 1(c)). Several routing protocols were proposed to support at least one IoT traffic pattern (e.g., eXtended Collection Tree Protocol (XCTP) [5], CodeDrip [6], and Mobile Matrix [3]). IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [4] became *de facto* the standard routing protocol for IoT. RPL is a distance vector protocol that builds a Destination Oriented Directed Acyclic Graph (DODAG).

[1]Note, there are others approaches to build P2M as presented in [2]–[4]



(a) Multipoint-to-Point routing structure.

(b) Point-to-Multipoint route structure and control messages.

(c) Data traffic patterns over routing structures.

Fig. 1. IoT's common routing structures and data traffic patterns.

RPL introduces some control packets in order to build routes: i) DODAG Information Object (DIO) advertises information allowing nodes to attach to DODAGs instances and obtain its configuration, initially the root fires the first DIO triggering a DODAG formation; ii) DODAG Information Solicitation (DIS) is used by a node to ask for valid DODAG instances nearby in its neighborhood, thus a DIS reception causes DIO transmissions; iii) DODAG Destination Advertisement Object (DAO) propagates destination information towards the root (through selected parents from the destination to root) intended to build P2M routes.

Most of IoT routing protocols (e.g., RPL and Mobile Matrix) relies on two key components to construct and maintain their routing structures: a routing metric and a timer scheme. The former, it is focused on expressing the link quality. Well known IoT routing metrics such as ETX and 4-Bit are useful to catch the highly dynamic and busty behavior of wireless technologies employed (e.g., IEEE 802.15.4) [3], [4], [7], regularly the metric is advertised into control packets (e.g., DIOs) helping nodes to find best paths. The focus of this work is on the timer schemes which concerns the firing interval between consecutive control packets advertisements.

III. Related Work and Problem Statement

Mobility is a major factor present in everyday life, thus the mobility of "things" is a natural event in the cyber-physical space. In IoT, even reduced device mobility can cause topology changes due to, for instance, the short-range wireless link technologies (e.g., IEEE 802.15.x). In such a situation, the routing protocol must rebuild routes to reflect the new topological organization as soon as possible mitigating network disconnections. Protocols advertise control packets from time to time to manage the routing structure. In RPL, an node can re-attach to a DODAG upon receiving a DIO (upwards routes) and then it transmits DAOs to build downwards routes.

The timer scheme governing the control advertisements plays a fundamental role to achieve the desired network performance. However, it faces a basic trade-off. If it frequently schedules advertisements, i.e., a small interval between consecutive beacons, it can quickly respond to topology problems at a cost of energy and channel overhead. On the other hand, if a timer scheme occasionally (large interval) advertises control packets, devices use less energy and bandwidth, but topological inconsistencies will persist for a long time.

In the literature, few attempts have been made to fill this gap, especially in IoT environments where mobility is present. The vanilla approach periodically sends advertisements in a fixed interval. However, the scheme leaves to the trade-off control to the network operator. Often, protocols use the adaptive beaconing Trickle Timer algorithm [8]. TT advertises control packets faster when topological inconsistencies occur in the routing structure, otherwise, it decreases the advertisement rate exponentially. The RPL specification suggests TT with 2.3 h as maximum period [4]. Therefore, a node can wait a long time before sending a control packet, which turns the tracking of topology problems an issue to standard RPL.

Reverse Trickle Timer algorithm [9] works in an opposite fashion than TT. It starts with a large interval between advertisements, then it halves the interval after each advertises fired. The authors argue that RevTT is suitable for mobile RPL networks where a mobile node attaches to a parent (preferably static), then it likely remains connected to this parent for a long time. As long as the node remains connected to the same parent, it is likely to that node moves away. This justifies a large initial interval and the short interval after long periods.

Besides those approaches, there are some proposals in the literature that make hard assumptions (e.g., knowledge about parent position) to operate [2], [10]. Also, vehicular networks is another field witch widely explore the benefits of adaptive beaconing approaches [11]–[13]. Those proposals focus on increase delivery rate and reduce the channel overload which is important for safety applications and routing protocols. However, they were not proposed to constrained IoT' devices.

Different from previous proposals, we do not propose a new timer scheme. Instead, we propose a selector to set a custom timer scheme given the mobility pattern of an IoT device. We advocate that by matching timer schemes to well-known mobility patterns, it is possible to balance the posed trade-off.

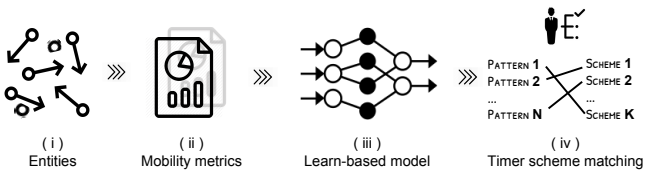


Fig. 2. Dribble pipeline.

IV. Dribble: the timer Selector Design

In this section, we detail Dribble, a learn-based timer scheme selector that helps IoT routing protocols to manage the mobility. Dribble learns the devices mobility patterns and then it assigns a customized timer scheme to each node in a distributed fashion. We have designed Dribble to be as generic as possible. Its method can be applied in a range of network topologies and sizes, though we have evaluated it to small scale IoT network such as a campus or building environments (see Sec.V).

Fig. 2 shows the Dribble pipeline process. It consists of the following stages: i) *Entities* (mobile or not) that Dribble intends to learn they mobility behavior; ii) *Mobility Metrics* that allow to Dribble to sense entities individually and use such information as input in the next stage; iii) *A learn-based model* (e.g., a neural network) which aims to classify the entity mobility pattern; iv) *The time scheme matching* in which, a specialist network operator manually matches the available timer schemes to a proper mobility pattern;

A. Entities

"Things" or just entities are the generic names referring to the devices employed in the IoT. Entities are of different types (e.g., TVs, home cleaners, vehicles, smartphones, etc.). Also, entities have a context (e.g., a place, a state, etc.). Here, we are interested in the mobility context. Although mobility can be studied in a broad spectrum, we just kept the mobility context simple by classifying entities into having mobility or no. For those entities that have mobility, we divide them into human-like and non-human mobility behavior. The context of the entities is obtained through sensing elements that also have a wide spectrum [14], e.g., sensors like GPS or accelerometer. Aware of the entities' context it is possible to design custom solutions as Dribble proposal.

B. Mobility Metrics

By using sensing elements, it is possible to study the entities mobility patterns. To do that, mobility metrics can be applied over the data caught from sensors. In [15], [16], the authors present and classify mobility metrics into spatial, temporal, and social classes. Spatial and temporal metrics describe how an entity behaves in the space domain and its rhythm over the time. Social metrics can describe the relationship (connectivity) between other entities.

In order to develop Dribble selector, we deliberately choose four metrics to capture temporal, spatial, and social properties from entities: i) Speed: it is the average node speed; ii) Travel Distance (TD): it is defined as the distance traveled between two consecutive locations; iii) Visit Time (VT): it is the time spent in each location visited by an entity; iv) Contact Entropy (CONEN): it measures the diversity of contacts of an entity based on Shannon's entropy. Entities with high CONEN value meet several other entities frequently and almost homogeneously. Otherwise, low CONEN values imply in low diversity and dissimilar meetings. These metrics were chosen due to their simplicity, thus IoT's commodity devices with some sensors (e.g., GPS or accelerometer) are able to compute them. For more mobility metrics details see [15], [16].

C. Learn-based Model

The next stage of Dribble's pipeline is to learn, through sensed data, the entities mobility context: static and mobile (human-like and non-human). Here, we have a classical classification or clustering problem. Although we have tested different models ranging from supervised to unsupervised, we focused on supervised models since we can obtain labeled data. We choose the Multi-Layer Perceptron (MLP) classifier as learning algorithm [17]. MLP learns a function $f(\cdot) : R^m \rightarrow R^p$, where m is the mobility metrics and p are the mobility patterns. One benefit of MLP is that it can learn a non-linear function for classifying more complex mobility context. Concerning applicability in IoT devices, our experiments show that a basic MLP can be applied to predict accurately the nodes mobility patterns.

D. Timer Scheme Matching

The timer scheme matching is a key stage of Dribble. Traditionally, just one timer scheme governs the entire IoT network, which is a hard-assumption, especially if the network entities present mobility. However, in Dribble, the main goal is to set a specific timer scheme for each entity conveniently. This makes sense since entities have different roles, capabilities, and behavior. Thus, they should have different timer schemes.

Currently, Dribble relies on a specialist to make the matching between mobility patterns and timer schemes. To do that assignment, the specialist needs to understand the mobility pattern, and which is the timer scheme more suitable. However, this is a hard task, thus to mitigate this burden, the network specialist can use the simulation methodology applied in Sec. V to support its decision. Also, it is expected that entities change their mobility behavior over time. For example, humans sometimes behave like a static entity (e.g., in an office), sometimes as a mobile entity (e.g., moving in a grocery). Dribble can re-evaluate the entity mobility pattern by just redoing its process. However, the network operator must set when the re-evaluation should be done.

V. Evaluation

In this section, we describe Dribble performance evaluation against the baseline Periodic fixed timer scheme, Reverse Trickle scheme as well as the widely used Trickle algorithm. Also, we present Dribble intermediate stage results.

TABLE I
DEFAULT SIMULATION PARAMETERS

Simulation setup	
Duration	15 days
# of Nodes	200
Base station	1 center
Distribution	Random
Radio range	200 m UDG
DIM	1500 m X 1500 m
# of random topologies	15
Transmission Model	CC2420-like
Timer schemes	
Trickle and Reverse Trickle	min = 2 s, max = 1024 s
Periodic	90 s

TABLE II
MOBILITY MODELS PARAMETERS

GRM-MIT		CRWP	
Group Duration	720 h	Node Speed (max)	5 m/s
Path time	300 s	T_{pause} (const.)	600 s
Statistical parameters		# Stops	Unif(0, 10)
α_{gmt}	2	PerMobNodes	50 %
β_{gmt}	720		
α_{dur}	2		
β_{dur}	720		
α_{size}	2.24		
β_{size}	30		

The evaluation process was conducted on Sinalgo simulator [18] and we use the standard RPL as routing protocol [4]. Our RPL implementation has the three data traffic patterns (M2P, P2M, and P2P) enabled, storing and no-storing modes as well as hop-count and ETX as Objective Functions. The RPL specification does not define how and when DIS packets should be sent and also how an unreachable parent should be removed from the preferred parent set. Those mechanisms play a significant role to react upon topology changes, especially derived from mobile nodes. In this sense, we implement a simple but not the most efficient alternative. After a node attaches to a parent, it waits for a small acknowledgment from the parent for every DIO sent. Then, if a node is not acknowledged, then it purges the parent from preferred parent set and enters in floating DODAG state [4]. Next, it triggers DIS packets try to re-attach to a valid DODAG. In mobile scenarios, authors have recommended that mobile nodes should prioritize static nodes as a parent [2], [3], [9], we also implement this feature.

Table I lists the default simulation environment parameters. Our simulation ran in random network topologies composed of 200 nodes. In all topologies, one fixed node represents the base station positioned in the center of the field, 50 static nodes were distributed in a grid fashion, representing the infrastructure. Moreover, there are 149 mobile nodes being that 100 present human-like mobility pattern and 49 present non-human patterns. Following, the mobility patterns are described.

A. Entities Mobility Modelling

To the best of our knowledge, there is a lack of real and diverse mobility traces for IoT's entities, usually due to privacy-related or technical issues. To overcome such situation, researchers have developed mobility models to fill this gap [19], [20]. Those models do mimics of real mobile entities behavior allowing us to generate variable traces in terms of space, time, and size.

We employ mobility models in our simulated environment classifying them into two groups: human and non-human patterns. For human mobility model, we highlight the Small World In Motion (SWIM) [21] and the Group Regularity Mobility model (GRM) [22]. SWIM produces synthetic traces with similar properties of real mobility traces. It assumes that humans go to places near their home, where they meet others, and, eventually, they return to their homes. GRM presents similar properties, but it introduces the dynamics of group meetings and social community structure.

Over the years several non-human mobility models were proposed [16], [20], [23]. We highlight the Random Waypoint Mobility Model (RWP), a well-known mobility model to evaluate MANET routing protocols [23]. In RWP, the entities move freely in a random direction, velocity, and acceleration. Also, there is an RWP extension named Cyclical Random Waypoint Mobility Model (CRWP) [3], where the entities behave similarly as in RWP. However, for CRWP entities, after n chosen destinations, the mobile entity returns to its initial position. CRWP is useful to model scenarios where some entities move to different destinations, and eventually, they return to their initial positions, which is the case of objects (e.g., portable devices, environment cleaners, etc.) that move in homes, offices, universities, hospitals, factories, etc.

We use GRM and CRWP as human and non-human mobility patterns, respectively. Table II lists the model's parameters. For GRM, we set the parameters to reproduce MIT real trace [22] behavior. The statistical parameters are from truncated power laws with cut-off where α_* is the power law exponent and β_* the cut-off value: α_{gmt} and β_{gmt} define the group meeting times distribution parameters; α_{dur} and β_{dur} characterize the time that a group of entities will spend together. Finally, α_{size} and β_{size} define which entities will be at each group meeting. For more parameter's details see [22]. CRWP has four parameters [3]: i) *Speed:* speed which the mobile entity moves; ii) T_{pause}*:* the amount of time that the entity stays in a destination position; iii) *Stops:* number of stops that the mobile entity do before returning to its original position; iv) *PerMobNodes:* maximum percentage of entities that are out from its initial position in each instant of time.

B. Measuring Mobility

In our simulation environment, there are static and mobile entities classes. We used mobility metrics as discussed in Sec IV-B to capture the mobility patterns. MOCHA [15] and BonnMotion [16] are freely available tools utilized to extract the mobility metrics from the traces. Fig. 3 shows three mobility metrics analyzed for one of our simulated topologies. Static nodes present no speed and Travel Distance

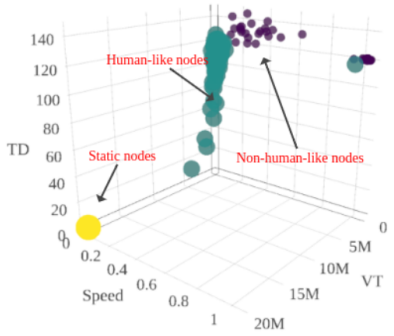


Fig. 3. Mobility metrics for each entity.

(TD), however, the Visit Time (VT) is high. While human-like nodes present high variability in TD with moderate VT. The non-human nodes present high-speed variability and high values of TD. Visually, it is possible to see three groups in our dataset, and this insight drove us to the learning-based classification method.

C. The Neural Network

We employ a Multi-Layer Perceptron to leaning the mobility context classes (static, human-like, and non-human). The MLP architecture (see Table III) was kept simple aiming to be suitable for constrained devices. The model was trained with mobility metrics measured from 10 random topologies composed by 200 nodes each. As model validation, we used 10-fold cross-validation over the data. Moreover, Fig. 4 shows the confusion matrix and Table III lists values of precision, recall, F1-score, and support. The results show high precision and recall, this is due to the target classes present disparate mobility metrics characteristics. Therefore, the model fits and predict correctly the entities classes.

D. Assigning Timer Schemes to Mobility Patterns

In our experiments, we assign the timer schemes to mobility pattern as follows: i) Non-human mobile entities were assigned to the Periodic scheme since they present the highest speed variability and travel distance. Therefore, a Periodic scheme with suitable short interval can better capture the mobility behavior. ii) Human-like mobile entities were assigned to RevTT. Those entities presented moderate VT and wide variability in TD. This suggests that entities usually arrived at the destination, stop for a while and then move again. This matches with RevTT proposal (recall Sec III). iii) Static entities were assigned to TT scheme since they represent infrastructure without any mobility, thus TT offers low control overhead when nodes experience network stability which usually occurs for infrastructure devices.

E. Simulation Results

To compare Dribble against single timer schemes, we use five remaining random network topologies. In each following plot, the bars or points represent the average, and the error bars indicate the confidence interval of 95 %.

TABLE III
MODEL PARAMETERS AND CLASSIFICATION REPORT

Neural Network Architecture and parameters				
Architecture	1 Hidden layer with 100 neurons			
Activation	Rectified linear unit function			
Learning rate	Constant			
# epochs	500			
Weight optimization	Adam			
Train dataset	Mobility metrics from 10 random topologies			
Validation model	10-fold cross-validation			
	Precision	**Recall**	**F1-score**	**Support**
Non-Human	1	0.99	0.99	165
Human	0.98	1	0.99	317
Static	1	0.96	0.98	171
avg / total	0.99	0.99	0.99	653

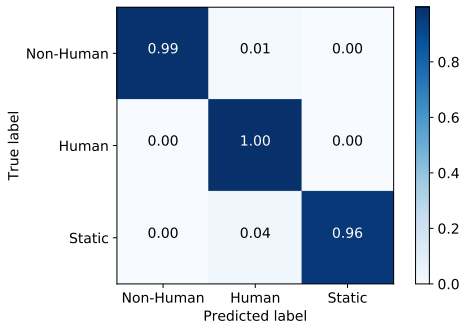


Fig. 4. Confusion matrix.

Firstly, we analyze the trade-off between control advertisement overhead and the average disconnection time along the 15 days of simulation in Fig. 5. In the graphic, it is desirable low control overhead and short disconnection timer. A low number of control packets implies in less energy expenditure and low channel occupancy. The disconnection time is the time spent from the moment that a node moves out from the parent radio range until it finds out a new parent. As expected, Periodic is the fastest timer scheme to find topology problems but it fires more control packets. On the other extreme, RevTT is more economical in terms of control overhead but topology problems persist for a long time. TT shows moderate trade-off balance. Dribble shows the better trade-off balance by quickly reacting to topology changes and triggering fewer control packets. Which can be explained by the customized timer scheme assignment to each node.

Owing to the wireless channel being a shared medium. When a node sends a message many neighbors nodes may overhear the transmission, even if the message is not intended to them resulting in unnecessary energy waste. Fig. 6(a) shows the overhearing transmissions (for all control flow DIOs, DAOs, and Acks) for each timer scheme. RevTT presented the lowest overhearing average, followed closely by Dribble and TT, while Periodic presented the highest overhearing average.

The average time of a set of nodes in a floating state[2] is

[2]A grounded DODAG offers connectivity (route towards the border router) to hosts, while a floating DODAG does not. It only provides routes to nodes within the floating DODAG [4].

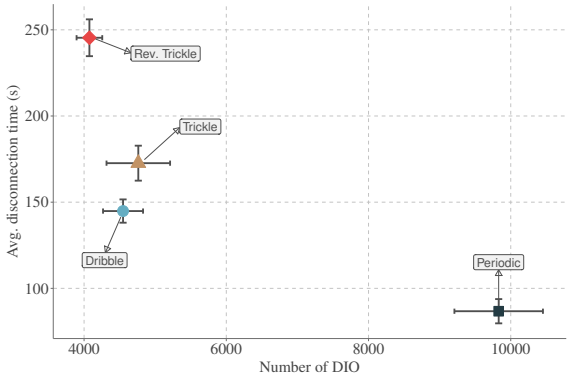
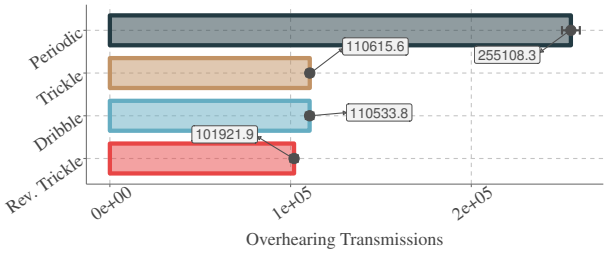
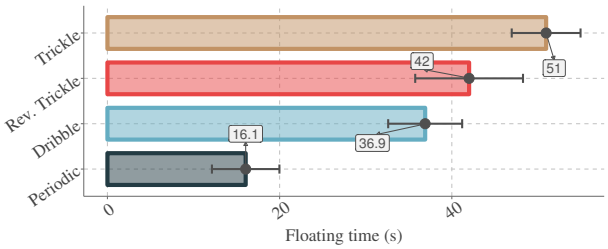


Fig. 5. The trade-off between control overhead and disconnection time.



(a)



(b)

Fig. 6. The control flow overhearing and the average time in a floating state.

shown in Fig.6(b). Dibble has a lower time in a floating state than TT and RevTT due to the mix of timer schemes running concurrently. Therefore, it is more likely to a node receive a control packet and fix the topology inconsistencies. Periodic presented the lowest time in the floating state, however, it uses more control advertisement as shown in Fig. 5.

VI. Conclusion

In this work, we proposed Dribble, a learn-based timer scheme selector, to improve the way timer schemes are used in IoT. Until now, routing protocols have used a single timer selector chosen without considering the entities' mobility behavior. Dribble goes further by setting a custom-made timer scheme to proper devices given their mobility pattern. We evaluate Dribble against Trickle Timer and Reverse Trickle Timer. Dribble presented a better trade-off balance between quick response to topology problems and energy expenditure and channel occupancy.

As future work, we aim to extensively improve Dribble to support fine-grained mobility contexts, better recursive re-evaluation of nodes mobility behavior and beaconing scheme matching. Also, provide an automatic way to associate mobility patterns to timer scheme, avoiding the specialist mediation.

Acknowledgments

We thank the research agencies CAPES, CNPq, FAPEMIG, and grants #15/24536-2 & #15/24494-8, São Paulo Research Foundation (FAPESP).

References

[1] O. Iova, P. Picco, T. Istomin, and C. Kiraly, "RPL: The Routing Standard for the Internet of Things... Or Is It?" *IEEE Communications Magazine*, vol. 54, no. 12, pp. 16–22, 2016.
[2] A. Oliveira and T. Vazão, "Low-power and lossy networks under mobility: A survey," *Computer Networks*, vol. 107, pp. 339–352, 2016.
[3] B. P. Santos, O. Goussevskaia, L. F. Vieira, M. A. Vieira, and A. A. Loureiro, "Mobile Matrix: Routing under mobility in IoT, IoMT, and Social IoT," *Ad Hoc Networks*, vol. 78, pp. 84 – 98, 2018.
[4] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, 2012.
[5] B. P. Santos, M. A. Vieira, and L. F. Vieira, "eXtend collection tree protocol," in *IEEE WCNC*, 2015, pp. 1512–1517.
[6] N. d. S. R. Júnior, M. A. M. Vieira, L. F. M. Vieira, and O. Gnawali, "Codedrip: Data dissemination protocol with network coding for wireless sensor networks," in *European Conference on Wireless Sensor Networks*. Springer, 2014, pp. 34–49.
[7] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks," *ACM TOSN*, vol. 10, no. 1, p. 16, 2013.
[8] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *USENIX NSDI*, 2004, pp. 2–2.
[9] C. Cobarzan, J. Montavont, and T. Noel, "Analysis and performance evaluation of RPL under mobility," in *IEEE ISCC*, 2014.
[10] F. Gara, L. Ben Saad, E. Ben Hamida, B. Tourancheau, and R. Ben Ayed, "An adaptive timer for RPL to handle mobility in wireless sensor networks," *IWCMC 2016*, no. 978, pp. 678–683, 2016.
[11] R. K. Schmidt, T. Leinmuller, E. Schoch, F. Kargl, and G. Schafer, "Exploration of adaptive beaconing for efficient intervehicle safety communication," *IEEE Network*, vol. 24, no. 1, pp. 14–19, 2010.
[12] C. Sommer, O. K. Tonguz, and F. Dressler, "Adaptive beaconing for delay-sensitive and congestion-aware traffic information systems," in *IEEE VNC*, 2010.
[13] A. Hassan, M. H. Ahmed, and M. A. Rahman, "Adaptive beaconing system based on fuzzy logic approach for vehicular network," *IEEE PIMRC*, pp. 2581–2585, 2013.
[14] A. Boukerche, A. A. Loureiro, E. F. Nakamura, H. A. Oliveira, H. S. Ramos, and L. A. Villas, "Cloud-assisted computing for event-driven mobile services," *Mobile Networks and Applications*, vol. 19, 2014.
[15] F. de Souza, A. C. Domingues, P. Vaz de Melo, and A. A.F. Loureiro, "MOCHA: A tool for mobility characterization," in *MSWiM*, 2018.
[16] N. Aschenbruck, R. Ernst, E. G. P., and M. Schwamborn, "BonnMotion: a mobility scenario generation and analysis tool," in *ICST*, 2010.
[17] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen, "Backpropagation Algorithm," http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm, 2011, (Accessed on 10/11/2018).
[18] E. D. C. Group, "Sinalgo-simulator for network algorithms," 2008.
[19] A. Hess, K. A. Hummel, W. N. Gansterer, and G. Haring, "Data-driven human mobility modeling: A survey and engineering guidance for mobile networking," *CSUR*, 2016.
[20] V. F. Mota, F. D. Cunha, D. F. Macedo, J. M. Nogueira, and A. A. Loureiro, "Protocols, mobility models and tools in opportunistic networks: A survey," *Computer Communications*, 2014.
[21] A. Mei and J. Stefa, "SWIM: A Simple Model to Generate Small Mobile Worlds," in *IEEE INFOCOM*, 2009.
[22] I. O. Nunes, C. Celes, M. D. Silva, P. O. Vaz de Melo, and A. A. Loureiro, "GRM: Group Regularity Mobility Model," in *MSWiM*, 2017.
[23] F. Bai and A. Helmy, "A survey of mobility models," *Wireless Adhoc Networks*, 2004.