# Adaptive Data Collection with Free Addressing and Dynamic Duty-Cycling for Sensor Networks

Fei Tong(✉) and Jianping Pan

University of Victoria, Victoria, BC, Canada
{tongfei,pan}@uvic.ca

**Abstract.** To improve the lifetime of the battery-powered sensors for data collection, duty-cycling is commonly adopted. A fixed duty cycle may cause a long packet delivery latency, low network capacity, and poor energy efficiency, especially in a frequently-reporting application. Moreover, considering a heterogeneous network consisting of various sensor platforms from different manufacturers, not only is node addressing with regard to address definition, management, and allocation difficult and costly, but also different addressing schemes will obstruct cross-platform communications. Based on the above considerations, this paper proposes an Adaptive Data Collection (ADC) with two features naturally and seamlessly integrated, i.e., free addressing and dynamic duty-cycling, to improve network heterogeneity, load adaptivity, and energy efficiency. ADC has been implemented in the Contiki Operating System. The evaluations based on a heterogeneous testbed consisting of two hardware platforms have demonstrated its practicality and efficacy.

**Keywords:** Sensor networks · Adaptive data collection · Dynamic duty-cycling · Free addressing · Testbed implementation

## 1 Introduction

Data collection is one of the most important applications of wireless sensor networks (WSNs), and can be widely found in environmental monitoring, infrastructure surveillance, scientific exploration, and other event monitoring applications [1]. Since radio is the most energy-consuming unit of a sensor node, and to improve the network lifetime of an energy-constrained WSN, existing data collection protocols commonly adopted or designed a duty-cycling scheme at the link layer [2–4], with which the radio of each node will periodically wake up and sleep according to an established time schedule. These data collection protocols usually employ a fixed duty cycle, in which the sleeping period could not be utilized for data transmission, even when the network traffic load is much beyond its capacity. However, a fixed duty cycle will cause unwanted energy consumption under light traffic loads, and network congestion and collision with packet retransmission or drop under heavy loads. This is because the accumulated data cannot be sent promptly just in the active period of the radio, which further


J.-H. Lee and S. Pack (Eds.): QShine 2016, LNICST 199, pp. 212–222, 2017.
DOI: 10.1007/978-3-319-60717-7_21

leads to a long packet delivery latency, low network capacity, and poor energy efficiency. Therefore, the conventional duty-cycled data collection protocols may not meet the real-time delivery requirement of the delay-sensitive applications.

Another concern about the data collection in WSNs is node addressing, e.g., the addressing for MAC/routing, which is often underestimated and even neglected in the prior data collection design. It is difficult and costly for the manufacturers of sensor nodes to assign a unique address for every node during the manufacturing phase, since there are several issues to be considered carefully, such as address definition, address space management, address allocation, etc. That is the reason all Zolertia Z1 [5] and Tmote Sky [6] motes use a default MAC address set by the manufacturers, so customers have to manually assign a unique MAC address to each node one by one for inter-node communication, which is quite inconvenient, especially when there are a large number of nodes to be deployed. In addition, a WSN may need different sensor platforms from different manufactures due to their different sensing capabilities. Different manufacturers, however, may have different addressing schemes, which will obstruct the cross-platform communications in a heterogeneous network consisting of various sensor platforms. On the other hand, the execution of an independent address allocation and exchange mechanism in runtime also causes a significant network overhead. Note that in a dense WSN, it is quite difficult to link the address of a node for communication with its location, and thus people assume the required location information can be determined by other means (e.g., GPS).

Based on the above considerations, this paper proposes an Adaptive Data Collection (ADC) protocol with free addressing and dynamic duty-cycling for multihop data collection in WSNs. Specifically, each node is identified for inter-node communication by using a Randomly-generated IDentifier (RID), plus its communication hop distance to the sink node. So there is no need to pre-assign a unique MAC address to each node or perform a routing address allocation and management procedure in the runtime of a network. Meanwhile, the sleeping period can be utilized on demand for data transmission to achieve a dynamic duty cycle adaptive to the network traffic load. Therefore, with the above two features, the network performance can be largely improved in terms of network heterogeneity, load adaptivity, and energy efficiency.

ADC intrinsically inherits from the previous designs several advanced features significant for multihop, duty-cycled data collection, such as pipelined forwarding so that a relaying node can forward a received packet in a short time to reduce the end-to-end packet delivery latency [4,7], inter-layer incorporation of network and link layers so that the protocol overhead can be reduced and the limited sensor node resources can be utilized efficiently [2–4], and lightweight schedule synchronization as an underlying support to the whole design [4]. Among these proposals, only the PDC protocol presented in [4] considers both the pipelined data collection and the underlying schedule synchronization over duty-cycled radios, practically and comprehensively. However, PDC does not consider the network heterogeneity, and traffic load adaptivity. In ADC, all the above features together with the two proposed in this paper (i.e., free address-

ing and dynamic duty-cycling) are naturally and seamlessly integrated and able to support each other by only relying on an RTS/CTS-like handshake, which is not only for data transmission as commonly utilized, but also for all other components in ADC. ADC has been implemented in the latest Contiki Operating System (OS) [8], a pioneering open-source OS for the Internet of Things. A testbed based on two real hardware platforms, i.e., Z1 [5] and MicaZ, forming a heterogeneous network, are established for performance evaluation.

The rest of the paper is structured as follows. ADC is presented in Sect. 2. The implementation and evaluation of ADC are shown in Sect. 3. Finally, Sect. 4 concludes the paper.

## 2 ADC Design

Similar to the design adopted by PDC [4], ADC only relies on an RTS/CTS-like handshake, called RTF/CTF (Request-to-Forward/Clear-to-Forward), to achieve all its features comprehensively. The difference lies in that ADC considers both free addressing and dynamic duty-cycling. There are three stages in ADC, including network initialization, topology establishment with free addressing, and data transmission with dynamic duty-cycling. In this section, we first have a general design description of ADC in Sect. 2.1. The three stages in ADC are then introduced in Sects. 2.2, 2.3 and 2.4, respectively.

### 2.1 General Design Description

During this stage, all nodes in ADC will be divided into different grades equivalent to their communication hop distances to the sink node (which is in grade 0). Meanwhile, each sensor node will maintain a periodic sleep-wakeup schedule defined as the radio status over time, according to which the radio of a node in grade $i$ ($i \in \mathbb{Z}$ and $i > 0$) periodically experiences three consecutive statuses, i.e., `R`: receiving a data packet from a sender in grade $i + 1$, `T`: transmitting a data packet to its receiver in grade $i - 1$, and `S`: sleeping. The schedules of any two adjacent grades along a path from the source to the sink node are staggered, so that a data received by a node during its `R` status can be forwarded consecutively during its upcoming `T` status and finally to the sink in a pipelined fashion, which can largely reduce the packet delivery latency and efficiently handle the traffic congestion by moving traffic quickly away from the congested area. Figure 1 shows the grade and schedule information maintained by the nodes along a path $A \rightarrow B \rightarrow C \rightarrow$ sink. Intuitively, `T` and `R` have the same maximum duration, which is defined as a slot for at most one transmission of a data packet with a maximum size set depending on a specific application. To stagger the schedules between any two grades so as to achieve the pipelined scheduling, the `S` duration is set to an integer multiple of the slot duration.

A data-gathering tree rooted at the sink node will be established in the second stage, where each node will build and maintain two neighbor tables, namely, Next-Hop Table (NHT) and Previous-Hop Table (PHT). Since there are

**Algorithm 1**: $wait(W)$

**Input**: $W$, $maxChildrenNum$
**Output**: $waitTime$
**if** *the current node is sink* **then**
  **return** $(waitTime = rand(W) \cdot \sigma)$;
**else**
  $w = W/(maxChildrenNum + 1)$;
  **return** $(waitTime = [w \cdot table\_length(PHT) + rand(w)] \cdot \sigma)$;
**end**

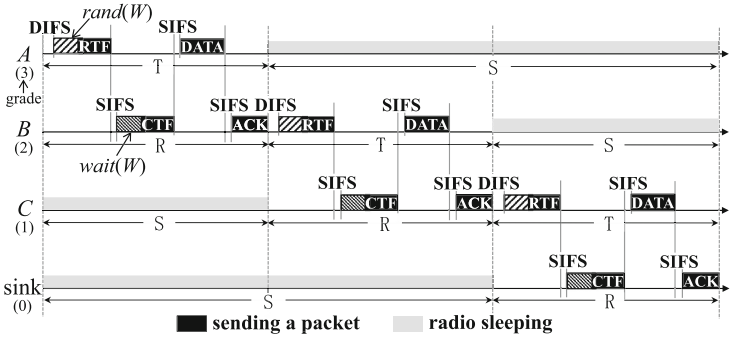



**Fig. 1.** An example used to show the ADC design.

no unique addresses pre-assigned to any nodes, free addressing will be achieved in this stage. A node's PHT is used to record the information related to its one or more children in the adjacent higher grade, including their RIDs and the schedule errors relative to them, while a node's NHT is used to record the information related to its only one parent node in the adjacent lower grade, including the parent's RID and the schedule error relative to the parent. In both tables, the schedule error information will be utilized for schedule synchronization. It is worth noting that there might be some other fields to be added for topology control and maintenance relying on the RTF/CTF handshake. For example, the number of children and residual energy of a node's parent (obtained from the CTF replied with by the parent) can be added to the node's NHT, which will be embedded in its RTF, so that a different lower-grade node with fewer children and/or more residual energy can make a comparison after receiving the RTF to determine to contend with the original parent for replying with a CTF. To increase the success probability of such a contention, the node will have a shorter waiting time, calculated by a function, $wait(W)$, as shown in Fig. 1, where $W$ is the maximum number of mini time-slots that the contention window of the handshake allows. Considering only the number of children a node has as an example, Algorithm 1 shows the design of $wait(W)$, where $maxChildrenNum$ is predetermined and represents the maximum number of children a non-sink node can have (the sink node has no such a limit), $\sigma$ is the time duration of one mini time-slot in the contention window, $rand(W)$ is the number of mini

time-slots randomly chosen within $W$, and *table_length*(PHT) is to obtain the number of items in a node's PHT, i.e., the number of children the node has.

Once the two tables are established, a node with pending data to be transmitted will initiate an RTF/CTF handshake with its parent in status `T` after waiting a randomly-chosen mini-time slots, $rand(W)$. If there is more than one packet, the `S` status will be utilized for data transmission to achieve a dynamic duty-cycling, as introduced in Sect. 2.4.

### 2.2 Network Initialization

During the network initialization stage, all non-isolated nodes will join the network. A node is thought to have joined the network if it has joined a grade and established a periodic sleep-wakeup schedule. The sink node initiates this stage by periodically broadcasting an RTF packet in its `T` status, as a periodic beacon, which was also typically utilized in the previous realistic implementations of sensor networks [2,4,9,10]. An RTF packet contains the sender's grade and status duration that the radio has stayed in status `T`. Any node which has not joined the network and thus keeps its radio on will determine its grade and schedule according to the received RTF. Then it also periodically broadcasts in its `T` status an RTF packet embedding its own grade and status duration. Finally all non-isolated nodes will join the network, as an example shown in Fig. 1. Note that at this stage, node address and CTF reply are not necessary.

### 2.3 Topology Establishment with Free Addressing

Once a node has identified its grade and schedule, it will leverage the periodic RTF/CTF *broadcast handshake* to establish a data-gathering tree for data collection. To avoid the cost and overhead of assigning a unique address to each sensor node during the manufacturing phase or performing an independent address allocation, exchange, and management mechanism in runtime, a node in ADC will use an RID for packet communication. Specifically, a node broadcasts an RTF embedding its RID. There might be several adjacent lower-grade nodes receiving the RTF and thus necessary to contend with each other for replying with CTFs with their RIDs embedded as well. The function $wait(W)$ ensures that those nodes with fewer children and/or higher residual energy reply first. The lower-grade node which successfully wins the contention will add the RTF sender into its PHT, and the RTF sender will add the CTF sender into its NHT after receiving the replied CTF. Then the two nodes will change from the broadcast handshake to *unicast handshake* for data transmission, as well as for schedule synchronization and topology control and maintenance.

Since RID is randomly generated, RID conflict is inevitable. It does not matter if two nodes in different grades have an identical RID, since their grades can be utilized to differentiate them. It may also happen that there are several nodes in the same grade having the same parent generated an identical RID, which will cause an RID conflict issue. Taking the communications among node 1, 2, and 3 in Fig. 2 for example, the following scheme is adopted in ADC to

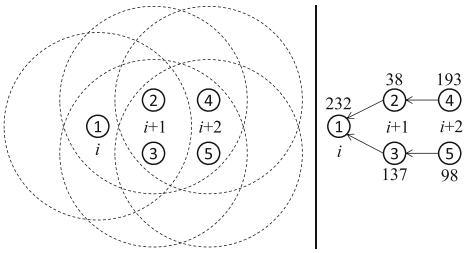



**Fig. 2.** Illustration of free addressing. For the left part of the vertical line, each dashed circle shows the node's communication range and the number inside each node shown as a solid circle is the node name, while for the right part, the number beside each node is its RID and the arrows indicate the network topology. Node 1 is in grade $i$ ($i \in \mathbb{Z}$ and $i \geq 0$), node 2 and 3 are in grade $i + 1$, and node 4 and 5 are in grade $i + 2$.

effectively solve the RID conflict between node 2 and 3. Assuming one of them, say node 3, fails in the broadcast RTF contention, then

- if node 3 can overhear the RTF from node 2, it will find the conflict and thus regenerate a new RID; otherwise,
- if node 3 can receive the CTF replied to node 2 from node 1, node 3 can also find the conflict and regenerate a new RID; otherwise,
- node 3 could not notice the conflict. Assume it wins the contention and broadcasts an RTF successfully next time. Every time a node receives a *broadcast* RTF, it will look up its PHT to find whether there is an RID identical to the one of the RTF sender. So node 1 notices the conflict after receiving the RTF from node 3, and generates a different RID embedded in its CTF for node 3.

Note that a node in ADC will not reply with CTF after it receives a broadcast RTF from any higher-grade node, until it determines its RID and parent in the lower grade. For example, node 3 will not be set as a parent by node 4 or 5 if it has not yet determined its RID and parent. In addition, it would be also possible that node 1 could successfully receive both the broadcast RTFs from node 2 and 3 if they could not hear each other during their contention (i.e., they are hidden terminals to each other). Node 1 will set a flag in its replied CTF to notify node 2 and 3 of regenerating their RIDs, if they have identical RIDs.

ADC only needs to handle the RID conflict among those nodes which have the same parent. Because, even if two nodes in the same grade have an identical RID, their data transmissions will not be affected as long as they have different parent. Therefore, ADC can use a small integer range for uniformly at random generating an RID for each node. In the ADC implementation in Contiki, we use an 8-bit variable for RID with a range of $[0, 255]$ (as shown in Fig. 2), since the number of the children a node has is usually much less than 256.

### 2.4 Data Transmission with Dynamic Duty-Cycling

The number of slots in status S determines how long a radio can sleep during a cycle. The more slots S contains, the lower duty cycle a node has and correspondingly the more energy can be saved. However, a low duty cycle will lead to a long packet delivery latency and low network throughput, especially under heavy traffic loads, while a high duty cycle will lead to an unwanted energy consumption, especially under light traffic loads. Therefore, a dynamic duty cycle adaptive to the network traffic load is highly expected. ADC can adaptively utilize the sleeping slots in status S to achieve dynamic duty-cycling (DDC).

To this end, the RTF packet contains a DDC flag, which will be set if a node has more than one packet to be transmitted, so that its receiver would wake up in status S instead of waiting for the next status R to receive the left packet. Depending on whether it is a last-hop transmission or not, the dynamic duty-cycling scheme in ADC will be discussed under two cases:

**Non-last-hop Transmission.** Figure 3 shows an example for illustration about the data transmissions along a path $A \rightarrow B \rightarrow C$, where node $A$, $B$, and $C$ are in grade $i+2$, $i+1$, and $i$ ($i \in \mathbb{Z}$ and $i \geq 1$), respectively. Assume node $A$ receives an RTF with the DDC flag set from a child node in grade $i+3$ in the R status before $t_0$ (not shown in the figure). Then node $A$ will adjust its schedule to reserve two continuous sleeping slots in status S (R and T in italic in the figure), which are used for receiving one more data packet from that child and forwarding the packet to its parent (node $B$), respectively. Meanwhile, in the upcoming T status after $t_0$, node $A$ will also set the DDC flag in its own RTF. So do node $B$ and $C$, and thus the data packet can still be forwarded in a pipelined fashion in status S. Considering the interference range is about two times longer than the transmission range for the wireless communication in an open-space environment [11], after any pair of statuses R and T, a node has to sleep at least two slots before waking up again for another data transmission, so that it will not interfere with its previous/next-hop neighbors. Therefore, the sleeping slots in status S cannot be utilized for data transmissions unless S contains at least 6 slots, and the maximum number of sleeping slots in status S that a node can utilize for data transmissions is $\left\lfloor \frac{\#\text{ of slots in S}-2}{4} \right\rfloor \times 2$.

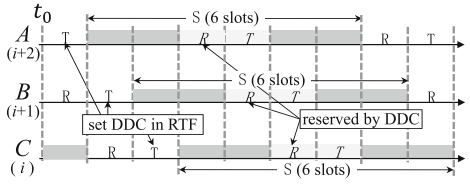
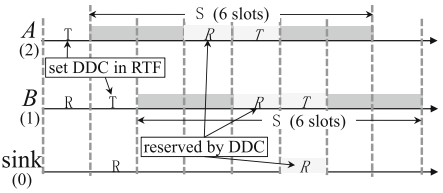


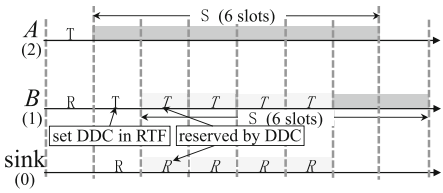

**Fig. 3.** Non-last-hop transmission along a path $A \rightarrow B \rightarrow C$.

**Last-Hop Transmission.** The data transmissions along a path $A \rightarrow B \rightarrow$ sink shown as in Fig. 4 are utilized for illustration. There are two cases for the

last-hop transmission, depending on whether or not a grade-two node (e.g., node $A$ in Fig. 4) either receives an RTF with the DDC flag set in status `R` or sets the DDC flag of its own RTF in status `T`. If so, it is similar to the non-last-hop transmission, as shown in Fig. 4(a), except that it is not necessary for node $B$ to set the DDC flag of its RTF packet to inform the sink node of waking up, since the sink node could be always awake for collecting data. Otherwise, all the sleeping slots of node $B$ except the two immediately before the next regular `R` status can be utilized for transmitting data, as shown in Fig. 4(b), since they will not interfere with the previous-hop transmissions.




(a) Node $A$ reserves the sleeping slots in status `S` for data transmission.




(b) Node $A$ has no data packet to be transmitted in status `S`.

**Fig. 4.** Last-hop transmission along a path $A \rightarrow B \rightarrow$ sink: (a) $A$ reserves the sleeping slots in status `S` for data transmission, and (b) $A$ has no reservation.

## 3 Implementation and Evaluation

ADC has been implemented in the latest Contiki OS [8] and evaluated in a testbed consisting of 5 Z1 and 2 MicaZ motes, as shown in Fig. 5. Contiki also provides a network simulator, called Cooja [12], which runs simulations based on the Contiki OS and fully emulated hardware devices, so that the codes developed for a Cooja simulation can be directly uploaded to real hardware devices even without any change, tremendously accelerating the development of realistic sensor networks. ADC is evaluated in terms of packet delivery ratio, average hop delivery latency, and average node duty cycle. PDC is used as a benchmark for comparison, which is one of the state-of-the-art collection protocols. Using the same code, we also conduct a Cooja simulation with 5 fully emulated Z1 and 2

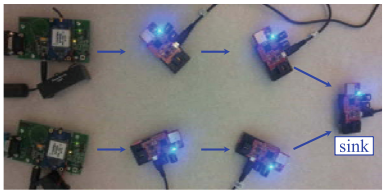
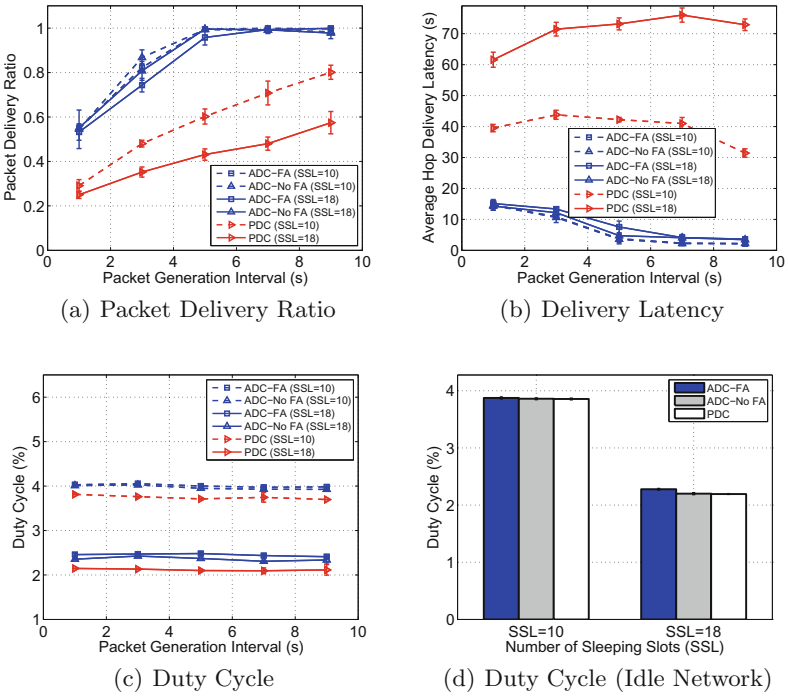



**Fig. 5.** A testbed consisting of 5 Z1 and 2 MicaZ motes forming a 3-hop network.

Tmote Sky (currently, Cooja does not support the communication between the emulated MicaZ and non-MicaZ platforms) based on the same network as in the testbed, and another one based on a larger heterogeneous network, which are not shown due to the page limit. All the results are consistent with each other.

In the testbed, there are six source nodes and one sink node, forming a 3-hop network with the topology shown as in Fig. 5. Each source node will generate 50 packets, with the packet generation interval (PGI) varying from 1 to 9 s. The retransmission limit for each packet is set to 5. The queue buffer size is set to 10 packets. A unique ID is pre-assigned to each node. Both the ADCs with and without free addressing are evaluated, denoted as "ADC-FA" and "ADC-No FA", respectively. In ADC-FA, nodes do not use the pre-assigned IDs but the RIDs generated in runtime. The number of sleeping slots (SSL) contained in `S` is set to 10 or 18, and thus for ADC, the maximum number of times that a node can wake up during its `S` status for data transmission is 2 or 4, respectively (i.e., at most $2 \times 2 = 4$ or $4 \times 2 = 8$ sleeping slots can be utilized for data transmission, respectively). We run each experiment 3 times for a duration of 40 min.

Since PDC cannot utilize the sleeping slots for data transmission, the packet delivery ratio in PDC is much lower than in ADC, as shown in Fig. 6(a). For PDC, a larger SSL will lead to a lower packet delivery ratio, because if a packet fails in being transmitted in the current cycle, it has to wait SSL slots for the




(a) Packet Delivery Ratio (b) Delivery Latency

(c) Duty Cycle (d) Duty Cycle (Idle Network)

**Fig. 6.** ADC performance evaluation based on the testbed shown in Fig. 5.

next cycle, and thus the node queue becomes full faster, resulting in more packet loss. In contrast, SSL affects ADC very slightly.

SSL also has a very slight effect on the average hop delivery latency in ADC, as shown in Fig. 6(b). However, with a larger SSL in PDC, a packet has a longer waiting time on average, leading to a larger delivery latency. In addition, as PGI increases, the packet delivery latency in PDC increases first and then decreases. Because as PGI increases, more packets can reach the sink node but with longer queue waiting time, which contributes to increasing the latency as a dominating part, while as PGI continuously increases, packets can be forwarded faster, which dominates and decreases the latency. In contrast, the packet delivery latency in ADC decreases always, and finally has almost no change when the queue waiting time of a packet cannot be decreased any more.

As shown in Fig. 6(c), with a larger SSL in PDC, a node can have more time to be in `S` status, corresponding to a lower duty cycle. ADC has a similar performance, because the least number of sleeping slots where a node in the ADC with SSL = 18 has to be asleep is larger than in the ADC with SSL = 10 ($18 - \lfloor \frac{18-2}{4} \rfloor \times 2 = 10$ *vs.* $10 - \lfloor \frac{10-2}{4} \rfloor \times 2 = 6$). In addition, the duty cycle in both ADC and PDC has almost no change as PGI increases, due to the fact that the number of packets to be transmitted is fixed. Since ADC can use the sleeping slots for data transmission, the duty cycle in ADC is a little higher than in PDC. As the network becomes idle, ADC adaptively does not wake up in `S` status and increasingly achieves the same duty cycle as PDC, as shown in Fig. 6(d).

For setting an appropriate SSL, obviously, there is a tradeoff in PDC among the three metrics. For example, a larger SSL can conserve more energy but lead to a lower packet delivery ratio and longer latency. On the contrary, a larger SSL is expected in ADC for conserving more energy, without obviously affecting the other two performance metrics. Furthermore, all the results in Fig. 6 show that the ADCs with and without free addressing have almost the same performance, which demonstrates the efficacy of the proposed free-addressing scheme.

## 4 Conclusion

This paper presented an adaptive data collection protocol, ADC, for WSNs. With the dynamic duty-cycling feature, ADC can utilize the sleeping slots adaptively for data transmission, improving the network load adaptivity and energy efficiency. With the free-addressing feature, there is no need to pre-assign unique addresses to sensor nodes or perform any address allocation and management procedure in runtime, improving the network heterogeneity.

**Acknowledgment.** This work is supported in part by NSERC, CFI, and BCKDF. The authors would also like to thank Dr. Kui Wu for his support on the testbed implementation.

## References


1. Wang, F., Liu, J.: Networked wireless sensor data collection: issues, challenges, and approaches. IEEE Commun. Surv. Tutor. **13**(4), 673–687 (2011)
2. Burri, N., Von Rickenbach, P., Wattenhofer, R.: Dozer: Ultra-low power data gathering in sensor networks. In: Proceedings of ACM/IEEE IPSN, pp. 450–459 (2007)
3. Ruzzelli, A.G., OHare, G.M., Jurdak, R.: MERLIN: cross-layer integration of MAC and routing for low duty-cycle sensor networks. Ad Hoc Netw. **6**(8), 1238–1257 (2008)
4. Tong, F., Zhang, R., Pan, J.: One handshake can achieve more: an energy-efficient, practical pipelined data collection for duty-cycled sensor networks. IEEE Sens. J. **PP**(99), 1–15 (2016)
5. Zolertia. http://zolertia.io/. Accessed 19 May 2016
6. Tmote Sky. http://tmote-sky.blogspot.ca/. Accessed 19 May 2016
7. Cao, Y., Guo, S., He, T.: Robust multi-pipeline scheduling in low-duty-cycle wireless sensor networks. In: Proceedings of IEEE INFOCOM, pp. 361–369 (2012)
8. Contiki OS. http://www.contiki-os.org. Accessed 19 May 2016
9. Gnawali, O., Fonseca, R., Jamieson, K., et al.: Collection tree protocol. In: Proceedings of ACM SenSys, pp. 1–14 (2009)
10. Werner-Allen, G., Lorincz, K., Johnson, J., et al.: Fidelity and yield in a volcano monitoring sensor network. In: Proceedings of USENIX OSDI, pp. 381–396 (2006)
11. Xu, K., Gerla, M., Bae, S.: How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks. In: Proceedings of IEEE GLOBECOM, pp. 72–76 (2002)
12. Osterlind, F., Dunkels, A., Eriksson, J., et al.: Cross-level sensor network simulation with COOJA. In: Proceedings of IEEE LCN, pp. 641–648 (2006)