Robust Multi-Pipeline Scheduling in Low-Duty-Cycle Wireless Sensor Networks

Yongle Cao, Shuo Guo and Tian He
Computer Science and Engineering, University of Minnesota
Email: {yongcao, sguo, tianhe}@cs.umn.edu

***Abstract*—Data collection is one of the major traffic pattern in wireless sensor networks, which requires regular source nodes to send data packets to a common sink node with limited end-to-end delay. However, the *sleep latency* brought by duty cycling mode results in significant rise on the delivery latency. In order to reduce unnecessary forwarding interruption, the state-of-the-art has proposed pipeline scheduling technique by allocating sequential wakeup time slots along the forwarding path. We experimentally show that previously proposed pipeline is fragile and ineffective in reality when wireless communication links are unreliable. To overcome such challenges and improve the performance on the delivery latency, we propose Robust Multi-pipeline Scheduling (*RMS*) algorithm to coordinate multiple parallel pipelines and switch the packet timely among different pipelines if failure happens in former attempts of transmissions. RMS combines the pipeline features with the advantages brought by multi-parents forwarding. Large-scale simulations and test-bed implementations verify that the end-to-end delivery latency can be reduced by 40% through exploiting multi-pipeline scheduled forwarding path with tolerable energy overhead.**

I. Introduction

Wireless sensor networks (WSN) is supposed to be used in a wide range of applications, such as habitat and environmental monitoring [1], target tracking [2] [3], scientific exploration [4], etc. In these applications, one common traffic pattern is *convergecast*, which is also known as many-to-one data collection. Under such traffic scenario, a set of sensor nodes conventionally send data packets to a common sink based on tree routing topology.

Additionally, large number of applications in WSN are also time-urgent. Those applications require the sink to have a snapshot of the network efficiently in limited time and thus strict delay constraint is usually imposed on the end-to-end packet delivery latency. However, in duty cycling network, such end-to-end delivery latency can be exacerbated significantly with the consequence of *sleep latency* [5] [6] that is introduced in schedule-based duty-cycling sensor network. In such network model, a sender has to hold the transmission and wait until the receiver wakes up based on its schedule.

In order to diminish the end-to-end delivery delay, certain techniques such as staggered wake up schedule [7], fast path algorithm [8], streamline schedule [9] are designed to wake up nodes along the data forwarding path at exactly the right time slots. All of those techniques present pipeline features to deliver data packets and ensure that each forwarding of a packet can catch up perfectly the wakeup time slot of the forwarder. However, most of such techniques only consider single predetermined route or fixed packet forwarder. This works fine only when all communication links are reliable and perfect, thus the end-to-end delay can be minimized. However, when the links becomes worse and unreliable, challenges arise and the delivery latency could increase significantly. Once a transmission fails, a node has to wait until the next time the specific forwarder wakes up, which indicates that the scheduled pipeline is essentially interrupted by unsuccessful transmissions. The situation can be even worse when the duty cycle is extremely low. This restriction is imposed by using only single and fixed parent in the data gathering tree. However, such restrictions could be removed if multiple parents are exploited to forward the packet. In this case, the route is actually a dynamic path instead of predetermined one.

Instead of unnecessarily persisting in waiting for a specific forwarder/parent to wake up, using multiple parents to forward the packet can decrease the sleep latency. Such idea originates from our experimental observation that *single scheduled pipeline* in the data gathering tree is always fragile due to the unreliable links. Therefore, we propose to switch the packet from one pipeline to another timely when failure is encountered in previous pipeline forwarding. Essentially, *multiple pipelines* are exploited to handle regular failure and interruption in pipeline forwarding. However, this alteration impacts the formation of staggered schedule of each pipeline. For single forwarder case, traffic all flows following a predetermined pipeline to decrease the delivery delay; while for multiple forwarders case, failures of transmission on one forwarder can make packet switch to another candidate forwarder and traffic thus change essentially from one pipeline to another. The allocation of wakeup slots has to be custom-designed to coordinate multiple pipelines and ensure the pipeline features are still maintained.

In this paper we attempt to design a robust multi-pipeline scheduling (RMS) algorithm which combines the staggered wakeup scheduling and the multi-parents forwarding scheme. First, RMS maintains the feature of pipeline so that it could minimize end-to-end packet delivery latency. On the other side, RMS coordinates multiple pipelines and utilizes the property of multi-parents data forwarding so that it could handle unreliable links and failures of transmission by timely switching packets among multiple pipelines. Specifically, the major contributions of this work are as follows:

- We experimentally reveal the frangibility of traditional single pipeline and illustrate that unreliable communi-

cation links could impair the effectiveness of pipeline scheduling in reducing delivery latency.

- To the best of our knowledge, this is the first work to propose a multi-pipeline scheduling protocol which considers the combined effect of unreliable radio links and the pipeline feature of packet forwarding. We further conduct large-scale simulations and realistic test-bed experiments to verify the effectiveness of RMS in reducing the end-to-end delivery latency in data collection.

The rest of the paper is organized as follows: Section II presents the related work briefly. Section III describes the motivation of our design. Section IV introduces the model and related assumptions. Section V gives the detailed design of multi-pipeline scheduling. Simulation and implementation results are provided in Section VI and Section VII, respectively. Finally, Section VIII concludes the paper.

II. Related Work

Due to the significance of energy efficiency in WSN, a bunch of scheduling algorithms have been investigated based on different objectives such as sensing coverage, network connectivity, throughput and etc. Another crucial category of scheduling algorithm is to minimize the delivery latency, especially in data collection. One of the most representative methodologies proposes to use pipeline feature in the scheduling. DMAC [7] is designed to allow continuous packet forwarding by giving the sleep schedule of a node an offset that depends upon its depth on the data gathering tree. It uses staggered wakeup schedules to create a pipeline for data propagation to reduce the latency of data collection. Cao et al. [9] proposes a similar technique denoted as streamlined wakeup. The idea is to synchronized duty cycles of nodes into a streamlined sequence to pipe the collecting data efficiently. Li et al. [8] investigate a fast path algorithm which provides fast data forwarding paths by adding additional wake-up periods on the nodes along paths from sources to sinks. However, all of the above works overlook the unreliable and unstable property of wireless radio channel and thus their efficiency could be undermined in the realistic environment.

As far as we know, no prior work has comprehensively considered maintaining both the pipeline feature of schedules and handling the regular transmission failures. We attempt to deal with this challenge by coordinating slots of different pipelines and switching packet among multiple cooperative pipelines.

III. Motivation

This section demonstrates two sets of experimental results collected from our indoor TinyOs/MicaZ test-bed. In our implementation, we observe the efficiency of traditional pipeline/streamline scheduling design in data collection. Specifically, we investigate the impact of link quality and duty cycle on the pipeline scheduling design in decreasing the end-to-end delivery latency.

A. Impact of Link Quality on Pipeline Scheduling

Representative works [7][9] on pipeline scheduling illustrate their superiority in decreasing end-to-end packet delivery latency. The network topology is organized as a tree when data collection is conducted. Each node has exactly one parent which helps to forward the packet to the upper level in the tree until the packet reaches the sink. Nodes on the routing path wake up sequentially to forward the packet to the next hop. However, these singly-pipelined scheduling designs ignore the fact that the link quality in reality is highly unreliable so that the announced efficiency could not necessarily be achieved when the link is imperfect.

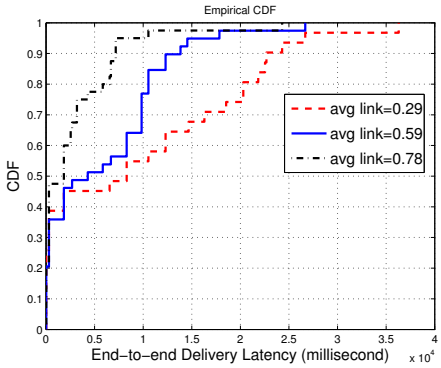


Fig. 1: Impact of Unreliable Link

To reveal this phenomenon, we construct a small scale of sensor network with 40 MicaZ motes. The sink is placed in the center of our test-bed and all other motes form a data gathering tree with depth of four. Each node in the network generates a packet designated to the sink. We record the end-to-end delivery latency for those packets received by the sink and the results are compared with varying link conditions. Fig. 1 shows that as the link quality become worse, the traditional pipelined schedule cannot guarantee small delivery latency in data collection since the pipeline is highly possible to be broken with bad communication links. For example, when the average link quality equals 0.78, 90% of nodes in the network have the delivery latency less than 7 seconds; while the same percentage of nodes has delivery latency less than 23 seconds if the average link equals 0.29.

B. Impact of Duty Cycle on Pipeline Scheduling

The impact of link quality on pipeline scheduling could be more severe when duty cycle goes lower. Another set of implementation results leads us to such conclusion. In this set of experiments, we change the duty cycle on each mote while maintaining similar link conditions. Fig. 2 illustrates the deficiency of singly-pipelined scheduling design could be much more evident when motes have lower duty cycle.

The rational is that once a transmission on the pipelined path fails, the packet forwarding has to wait until the next time the receiver wakes up. Lower duty cycle implies the interval between consecutive attempts of transmissions is larger and thus the delivery latency becomes greater.

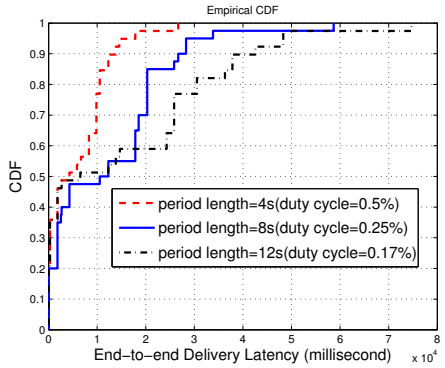


Fig. 2: Impact of Duty Cycle

IV. Model and Assumptions

This section first describes the schedule-based low-duty-cycle wireless sensor network model. After that, assumptions made in our design will be given.

A. Schedule-based Low-Duty-Cycle Network Model

A sensor node in low-duty-cycle network is in either active state or dormant state. Active nodes can transmit packets or sense and receive packets from neighbors. In order to wake up at a proper time instance to send out a packet, the sender should be aware of the time when the receiver is in active state. All sensor nodes maintain a local neighbor table to record working schedules for one-hop neighbors.

Sensor nodes determine their active/dormant states based on wakeup schedules. Since the schedule is normally periodic for sensing purpose, we use a circle to represent the time line of each working period as shown in Fig. 3. Although the length of a period varies depending on individual node, common length of period T can still be found (e.g., least common multiple of cycle for all nodes). T is further divided equally into multiple units, called time slots, in which sensor node is either active or dormant. To simplify, the length of each time slot is assigned equally to round-trip packet transmission time (including data and ACK). This assumption can be achieved by increasing time granularity. If multiple packets can be transmitted within single wakeup slot, multiple consecutive wakeup slots are regarded as being selected. Under such scenario, for node i, wakeup schedule Γ_i can be uniquely represented as a set of wakeup slots,$\Gamma_i = \{t_1^i, t_2^i, t_3^i, ..., t_N^i\}$, where t_j^i denotes the j^{th} wakeup slot for node i.

For example, in Fig. 3 where $T = 100$ is denoted as the length of a working period, node i is scheduled to wake up at time slot:$12, 30, 63$, then node i's wakeup schedule can be uniquely represented as $\Gamma_i = \{12, 30, 63\}$.

B. Topology Model and Traffic Model

Each node in the network is aware of its hop count away from the sink. Such hop count can be determined in the initialization phase. The sink starts to broadcast notification packets with hop count equaling 0. Then each node who receives such packets selects the minimum hop count as its updated value, adds one to the hop count and then broadcasts

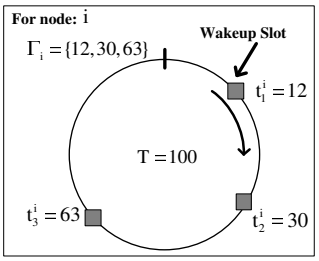
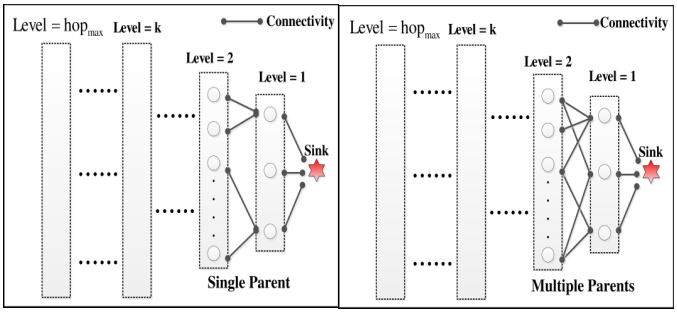


Fig. 3: Periodic Schedule

the packet to all its neighbors except the one where it originally comes from. This phase works similarly to breadth-first-search (BFS) until convergence is achieved. Nodes with hop count equalling k are named as level k node and finally a level-by-level network topology is generated, as showed in Fig. 4a.

In traditional data gathering tree, a level k node chooses only one of its neighbors in level $k-1$ as its parent and thus form a tree-based network topology. Fig. 4a gives a simple example of such data gathering tree. We note that node $i^{(k)}$ in level k is only connected to single node $j^{(k-1)}$ in level $k-1$, which is denoted as $parent[i^{(k)}] = \{j^{(k-1)}\}$. Our network topology in this paper is modified based on the data gathering tree model. Instead of being connected to only one node in the upper level, node $i^{(k)}$ in level k has multiple parents in level $k-1$ and thus we have $parent[i^{(k)}] = \{j_1^{(k-1)}, j_2^{(k-1)}, ...\}$. Fig. 4b is an example of our data gathering network topology modified based on corresponding data gathering tree.



(a) Traditional Data Gathering Tree (b) Data Gathering Topology

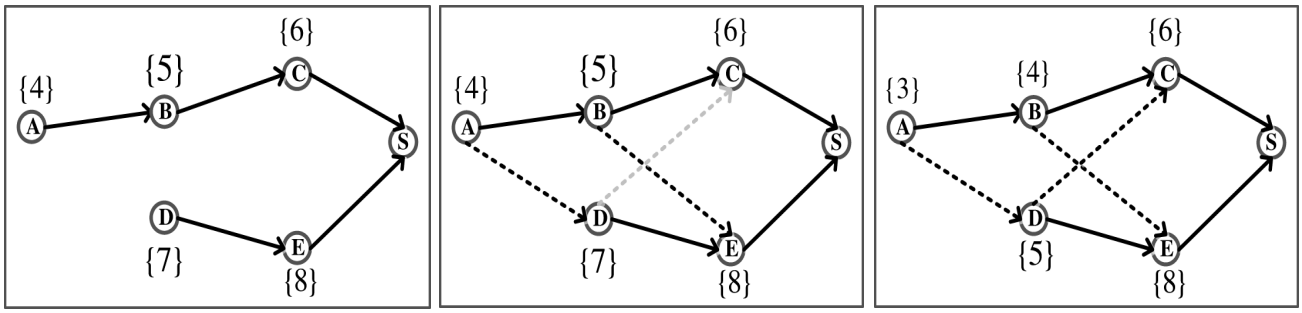
Fig. 4: Topology

The traffic model discussed in this paper is focused on unidirectional data traffic. The traffic pattern consists of data collected from regular nodes to a common sink. Such traffic is also the major traffic pattern utilized in the applications such as event detection, periodic sampling and etc. In conclusion, the data traffic flows through aforementioned data gathering topology until it reaches the sink. Our goal is to minimize the average packet delivery latency given this traffic model.

C. Assumptions

The following assumptions are made in our design:

- *Locally Synchronized.* The clocks on each sensor in the neighborhood are synchronized, which denotes that given the schedule of the receiver, a potential sender is aware of the time to transmit the packet. FTSP in [10] proposed to exchange a few bytes of packets among neighbors every 15 minutes to achieve clock synchronization accuracy as



(a) Pipeline Scheduling Tree (b) Multi-Parents Forwarding (c) Multi-Pipeline Scheduling

Fig. 5: Data Gathering Example

much as $2.24\mu s$. Since the period of one active time instance is normally $2000\mu s$ to $20,000\mu s$, FTSP can provide sufficient accuracy.

- *Unreliable and Measurable Links.* Wireless radio links between neighbors are imperfect, which means failures of transmissions always exist in the process of data collection. Besides, we also assume link qualities of neighbors can be measured. In practice, the 4-bit estimator proposed in [11] addresses link dynamics by actively using data packets and periodic beacons to measure link quality.
- *Forwarding Constraint.* A node is aware of neighboring hop count and only forwards the packet to nodes with smaller hop count. This assumption is used to avoid forwarding loop in data collection.

V. Main Design

A. Design Overview

Given data gathering tree, traditional pipeline scheduling algorithm requires wakeup time slots along the forwarding path to be consecutive. Fig. 5a shows a simple example in which there are two forwarding paths $A \to B \to C \to S$ and $D \to E \to S$. The schedule of each node is also pipelined as $\{4\} \to \{5\} \to \{6\}$ and $\{7\} \to \{8\}$. This works perfectly fine if no transmission failure happens in data collection. However, if node A fails to send the packet to B at $\{5\}$ and still insists on forwarding along the pipelined path, it has to wait until $\{105\}$ when node B wakes up again(suppose $T = 100$).

Instead of persisting on waiting for a single parent, Fig. 5b utilize multi-parents to help forward the packet so as to avoid unnecessary standby time. This method takes advantage of the latest transmission results to search a preferable route to forward the packet. As showed in Fig. 5b, after encountering the failure at $\{5\}$, node A decides to try D immediately. Once the packet arrives at D at $\{7\}$, it still can catch up the pipeline along $D \to E \to S$. Similar scenario could also happen on link $B \to E$. We note the introduction of link $A \to D$ and $B \to E$ successfully switches the packet among parallel pipelines and mitigates the influence of broken pipeline. However, we also take notice of link $D \to C$. Though node C is D's neighbor, D cannot catch up the wakeup slot of C if the transmission between D and E has failed. D has to wait until next period to try either node C or node E. The rational is that different pipelines have not been coordinated to cooperate with each other even packet can be forwarded by multiple parents.

Fig. 5c shows a possible better scheduling result which bridges the merits demonstrated in Fig. 5a and Fig. 5b. In this case, not only links $A \to D, B \to E$ but also link $D \to C$ can be fully utilized to switch the packet timely among pipelines. In our design, we attempt to combine the pipeline feature in Fig. 5a and the advantages brought by multi-parents forwarding in Fig. 5b. To achieve this, we need to coordinate parallel pipelines so that packet forwarding can be switched among different pipelines. Overall, RMS consists of three major steps:

- **Selection of Virtual Forwarding Set.** Each node decides its *virtual forwarding set* (refer to Section V-B) based on the link quality to its potential forwarders and predetermined one-hop delivery ratio. Packet could be forwarded opportunistically to any node resides in the virtual forwarding set. The selection of virtual forwarding set is to prepare for the further decision on scheduling.
- **Propagative Scheduling.** After having hop count ready, schedules of nodes are decided based on minimizing the expected delay of packet forwarding between two consecutive levels. The decision making phase propagates from the sink to other nodes in a level-by-level manner.
- **Overlapping Resolution.** Since each node makes scheduling decision distributively, a node could have multiple parents wake up at the same time slot. In order to fully utilize the opportunities to catch up potential pipelines, simultaneous wakeup slots of parents can be adjusted slightly and a technique called *On-the-fly Shifting* can be resorted to resolve such overlapping conflict.

B. Virtual Forwarding Set

Compared with merely using predetermined single routing path, our design exploits the property of opportunistic forwarding. In other words, the routing path is decided dynamically based on the timely transmission results. For a node $i^{(k)}$ in level k, its *forwarding set* includes any node in upper level who could be a potential forwarder so as to decrease the sleep latency of packet between level k and level $k-1$. Apart from forwarding set, each node also maintains a *virtual forwarding*

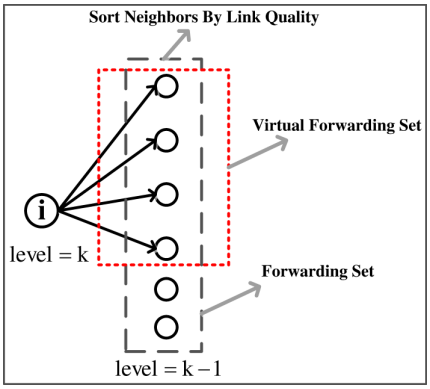


Fig. 6: Vitural Forwarding Set Vs. Forwarding Set

set, which is a subset of the forwarding set, for the purpose of multi-pipeline scheduling. The rational of introducing virtual forwarding set is that packets will be mainly (with high probability) forwarded by one of nodes in this set. As a consequence, the schedule of a child node should be primarily pipelined based on the schedule of nodes that are in its virtual forwarding set.

The virtual forwarding set could be decided based on the link quality to nodes in the forwarding set and predetermined one-hop packet delivery ratio which can be configured as a system parameter. Fig. 6 illustrates how the virtual forwarding set can be determined. First, node i sort the links to all neighboring nodes in the forwarding set, suppose the link quality is $\{q_{i1}, q_{i2}, ...q_{in}\}$ where $q_{i1} \geqslant q_{i2} \geqslant \cdots \geqslant q_{in}$. In order to minimize the expected sleep latency of a packet, node i is expected to choose better links to send packet first. Suppose the threshold of one-hop packet delivery ratio is given as ϕ, the probability that the transmission is successful at least once within the first k attempts is $1-(1-q_{i1})(1-q_{i2})\cdots(1-q_{ik})$, so in order to satisfy the threshold of one-hop delivery ratio, the first M best links are included in the virtual forwarding set of node i:

$$1 - \prod_{k=1}^{M-1}(1-q_{ik}) < \phi \tag{1}$$

and

$$1 - \prod_{k=1}^{M}(1-q_{ik}) \geqslant \phi \tag{2}$$

The selection of virtual forwarding set essentially affects the decision on picking which forwarder to pipeline. Note that the *virtual* forwarding set is used merely for scheduling purpose, once the schedule is fixed, each node follows the forwarding set to do the data forwarding. Also note that since the routing algorithm is to use every wakeup slot in the forwarding set to attempt retransmission, the size of forwarding set actually reflects the trade-off consideration between energy cost and delivery latency. In one extreme case, the forwarding set could contain all neighboring nodes in upper level, then the average energy cost could be significant since certain bad links are also used to try transmissions. Another extreme case is that the forwarding set is equivalent to virtual forwarding set, which means the size of forwarding set is exactly large enough to guarantee the one-hop delivery ratio requirement. This could avoid retransmissions with low success probability and thus save energy, however, it could also increases the standby time waiting until the next time a forwarder wakes up.

C. Propagative Scheduling

In this step, each node decides its wakeup schedule distributively with the purpose of minimizing the expected delay of packet between two consecutive levels leading to smaller end-to-end delay. After the virtual forwarding set has been determined in each node, the scheduling phase propagates in a level-by-level manner starting from the sink. For those nodes in the first level, the sink can hard-allocate the schedule. In this case, the sink is able to control the most suitable time when it expects the incoming of packets. For nodes in other levels, we derive a recursive and distributive scheduling algorithm. Since the scheduling phase propagates from sink to deeper levels, the core problem converts to how to decide a child's schedule given the schedules of its parents in the virtual forwarding set.

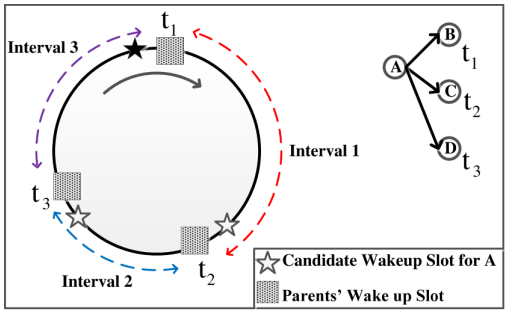


Fig. 7: Select Candidate Slot During Propagative Scheduling

Fig. 7 is an example used to explain how the recursive scheduling algorithm works. Node A has three parents in its virtual forwarding set, which have wakeup time slot at $\{t_1\}, \{t_2\}$ and $\{t_3\}$ respectively. At the first glance, it seems that the wakeup time slot of A can be arbitrary all over the period of schedule. However, we can prove that the optimal allocation of time slot for node A can only be selected from a limited candidate time set in order to minimize the expected delay of packet between two consecutive levels. For example, the pentagram in Fig. 7 represent the candidate time set $\{t_1-1, t_2-1, t_3-1\}$ for node A's optimal slot. We first give our conclusion and then prove it by contradiction.

LEMMA 1: For node i, given its parents' wakeup time slots $t_1, t_2, ..., t_m$, where $t_1 \leqslant t_2, ... \leqslant t_m$, in order to reduce the expected delivery delay of packet forwarding between node i and its upper level, the optimal time slot for node i can only be selected from a limited candidate time set $\{t_1-1, t_2-1, ..., t_k-1\}$.

Proof: Since the parents' wakeup time slots divide one period of schedule into several intervals, that is $t_1 \longrightarrow t_2, t_2 \longrightarrow t_3 ..., t_{m-1} \longrightarrow t_m$, the wakeup slot of node i (denoted as $t(i)$) must lie in one of these intervals, suppose $t(i)$ lies in the interval $t_k \longrightarrow t_{k+1}$. Note that the variable of delay from node i to its neighbors is a discrete random variable depending on the real transmission result, the value of

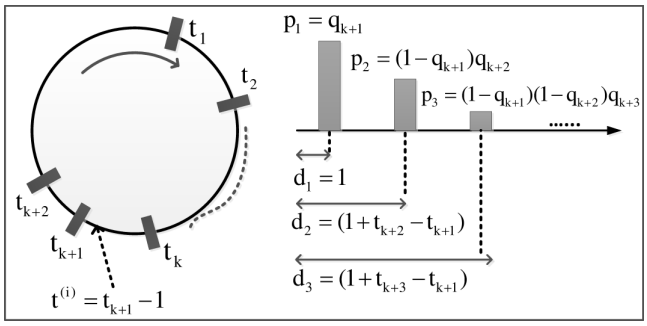


Fig. 8: Computation of Expected Delay

delay can be: $|t_{k+1}-t^{(i)}|_T, |t_{k+2}-t^{(i)}|_T, ..., |t_m-t^{(i)}|_T, |t_1-t^{(i)}|_T, ..., |t_k-t^{(i)}|_T$, where $|t_1-t_2|_T$ is the modular delay and is defined as

$$|t_1-t_2|_T = \begin{cases} t_1-t_2 & t_1 \geq t_2; \\ t_1+T-t_2 & t_1 < t_2. \end{cases} \quad (3)$$

Besides, suppose the probability mass function (pmf) of delay is $f_D(d)$. By contradiction, if the above lemma does not hold, which means $t(i) \notin \{t_1-1, t_2-1, ..., t_k-1\}$, then the expected delay between node i and its' upper level can be reduced further by adjusting $t(i)$. To show such decrease, we adjust $t(i)$ from it original position to $t_{k+1}-1$. Note that such adjustment happens within the same interval so that it would not change the pmf of delay since the sequence of transmissions maintains the same. However, the value of delay is now reduced to $|t_{k+1}-(t_{k+1}-1)|_T, |t_{k+2}-(t_{k+1}-1)|_T, ..., |t_m-(t_{k+1}-1)|_T, |t_1-(t_{k+1}-1)|_T, ..., |t_k-(t_{k+1}-1)|_T$. So such adjustment can decrease the expected delay of packets generated by node i, which contradicts our assumption. ■

In order to show how to select the best slot from the aforementioned candidate time set, we use Fig. 8 to illustrate the computation of expected delay for packet generated by node i to reach the upper level. In Fig. 8, $t_1, t_2, ..., t_m$ are the wakeup slots for node i's neighbors. Suppose node i select one of slot $t^{(i)} = t_{k+1}-1$ from its candidate set. Since a packet from node i (no matter the packet is generated by itself or forwarded from its children) can only generated at $t^{(i)}$, so node i would try to send the packet to neighbor $k+1$ first. Then the probability for such packet to be delivered successfully at the first attempt is $p_1 = q_{k+1}$, where q_{k+1} denotes the link quality from node i to neighbor $k+1$. If the first attempt fails, then node i would try neighbor $k+2$ subsequently. Generally, the probability $\Pr(n)$ that the packet transmission by node i fails at the first $n-1$ times while is successful at the n^{th} attempt is:

$$\Pr(n,k) = \prod_{j=1}^{n-1}(1-q_{k+j})q_{k+n} \quad (4)$$

where suppose node i selects neighbor $k+1$ to pipeline, that is $t(i) = t_{k+1}-1$.

Small probability event may happen when the first R_{max} all fails. In this case, the packet would be dropped by node i. Note that the probability for the packet is transmitted successfully at the n^{th} attempt is under the condition that the packet is delivered successfully within R_{max} attempts. The conditional probability can be represented as:

$$\Pr(n,k)_{Cond} = \frac{\prod_{j=1}^{n-1}(1-q_{k+j})q_{k+n}}{1-\prod_{j=1}^{R_{max}}(1-q_{k+j})} \quad (5)$$

Fig. 8 also shows the delay value for the packet to be received successfully. For example, delay to reach neighbor $k+1$ is $d_1 = t_{k+1}-(t_{k+1}-1)$; delay to reach neighbor $k+2$ is $d_2 = t_{k+2}-(t_{k+1}-1)$. Generally, the delay for the packet to be delivered successfully at the n^{th} attempt is:

$$d_n = t_{k+n}-(t_{k+1}-1) \quad (6)$$

Thus, if node i selects neighbor $k+1$ to pipeline which means $t(i) = t_{k+1}-1$, then the expected delay for a packet from node i to reach upper level is:

$$E_i(k) = \Sigma_{j=1}^{R_{max}} d_j \Pr(j,k)_{Cond} \quad (7)$$

Consequently, for node i, given the wakeup slots of parents, the candidate time set can be determined by lemma 1. However, due to the energy budget on the mote, one optimal slot needs to be selected from the candidate time set. Then based on Eq. 7, the expected delay for one candidate slot can be computed, thus one traversal of the candidate set is enough to find out the optimal slot which leads to minimum expected delay for a packet to reach upper level. The essence of choosing optimal slot from candidate set is to coordinate multiple pipelines and specifically, it select one parent in the virtual forwarding set to pipeline while ensuring the packet could be switched as promptly as possible to another pipeline if the former attempts fail. For the computation complexity, Equ. 7 can be accomplished with $O(R_{max}^2)$. Suppose the number of neighbors is bounded by a constant C, so the overall complexity for the iteration through candidate time set is $O(CR_{max}^2)$. Note that more candidate slots could be selected by node i depending on local support of duty cycle.

D. Avoid Simultaneously Wakeup Parents

1) Simultaneous Wake-Up: Section V-C has gone through the procedure of propagative scheduling. This section tackles some practical issues and presents optimization on propagative scheduling in a further step.

One phenomenon in our experiments shows that multi-parents of the same child node could wake up simultaneously, which may contradict the expectation to reduce sleep delay using multi-parents forwarding. Fig. 9a gives such an example. Node A's virtual forwarding set consists of two nodes, node B and node C, which wake up at the same time slot $\{5\}$. In this case, due to simultaneously wakeup parents, node A can only utilize one parent, either B or C, to help forward the packet. Once the transmission fails, A has to wait for the next working period. Whereas, if node B and node C could wake up at different time, then A would have two opportunities each period to catch up the pipeline. The reason inducing simultaneous wakeup problem is because node B and node C select their optimal time slot independently from their

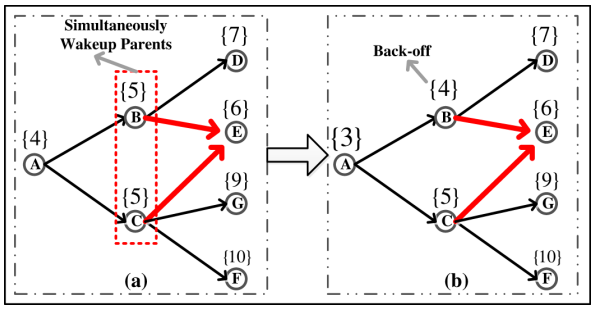


Fig. 9: Simultaneous Wakeup Parents

candidate time set, however, their candidate time set may share the same slot. For example, in Fig. 9a, both B and C select node E to pipeline and thus choose $\{5\}$ as their wakeup slots.

2) On-the-fly Shifting: In order to avoid simultaneously wakeup parents, we utilize a technique called *on-the-fly shifting*. This technique can shift simultaneous wakeup slots on-the-fly while maintaining the advantages brought by propagative scheduling. Take Fig. 9b as an example, due to simultaneous wakeup slots of B and C, node A selects one of them to forward the packet, say, C is selected. Then when the transmission from A to C is happening, since B is within the communication range, so B can actually overhear such transmission and discover the target id is not itself. As a result, B knows there must be another node which also wakes up at the same slot $\{5\}$. Then node B decides to back-off and shift its schedule one slot ahead to $\{4\}$ so as to remove the overlapping of wakeup slots with other nodes. Note that such back-off process may need continue if node B overhear another similar transmission again at $\{4\}$ and it would stop when no overlapping exits any more. Once a node finishes shifting, it need to advertise its children its new wakeup slot then its children can recompute their optimal wakeup slots. In the above example, node A is advertised and it reselects $\{3\}$ as its optimal slot.

VI. Simulation and Evaluation

In this section, we provide the performance results of our proposed RMS algorithm under numerous network settings. In order to show the efficiency of our design, we also compared RMS with two other baseline solutions:

- **Single Pipeline:** In order to show the fragility of traditional staggered wakeup scheduling (or streamline scheduling) and the efficiency of RMS in reducing the end-to-end delay, we implement single pipeline scheduling design in which each node has only one forwarder (with best link) and the wakeup slots along the forwarding path are pipelined as proposed in [7] [9].
- **RMS-Random:** Section V-C gives the design on the selection of wakeup slot from limited candidate set, which essentially coordinates the schedule among multiple pipelines. We also implement a second baseline design called RMS-Random which randomly selects wakeup slot from the candidate set instead of selecting the optimal slot based on computation given by Eq. 7.

A. Simulation Setup

We deploy a large number of nodes in a $200m \times 200m$ square field and randomly generate the network topology. The network size varies from 200 to 600 nodes and the sink is placed in the center of the field. The links among nodes are simulated according to the radio model proposed in [12]. For each simulation setting, statistical results are collected from 50 runs with different seeds. Each node sends 50 packets to the sink in the phase of data collection and average data is reported in the next section.

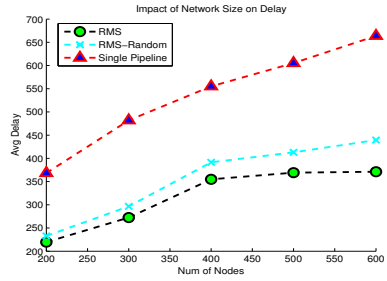
B. Performance Evaluation

This section compares end-to-end delivery latency, energy consumption per packet and packet delivery ratio among RMS, RMS-Random and Single Pipeline under different network configurations.

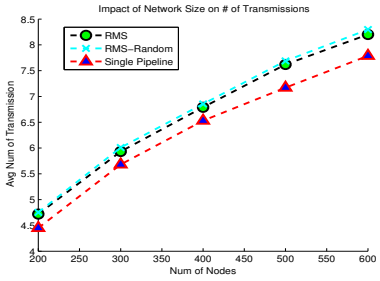
*1) **Impact of Network Size**:* In order to show the impact of network size and scale of data gathering topology on our design, we simulate RMS under different number of nodes and length of field. We change the total number of nodes from 200 to 600 while expanding the length of field from $140m$ to $245m$ to keep similar network density.

Fig. 10a shows that RMS reduces end-to-end delay by $40\% \sim 50\%$ compared with single pipelined scheduling design. Such reduction mainly results from two aspects: RMS 1) switches packet transmissions among multiple pipelines by utilizing multi-parents forwarding; 2) coordinates wakeup slots among multiple pipelines. Also, RMS can reduce the delay up to 23% compared with RMS-Random, which verifies the effectiveness of selection of optimal wakeup slot based on minimizing expected delivery delay among two consecutive levels in the data gathering topology. For all of these three designs, the end-to-end delay would increase as the data gathering topology grows. Fig. 10b reports that RMS has a little bit energy overhead compared with Single Pipeline. The rational is that Single Pipeline design could attempt to send packet only to the forwarder which has the best link, while for RMS, it might also try worse links in order to switch the packets among pipelines if a failure happens. So RMS actually reduces the end-to-end delay by adding a little bit energy overhead, which is admissible in time-urgent applications. We also note the delivery ratio can maintain above 90% for all three designs, as showed in Fig. 10c.

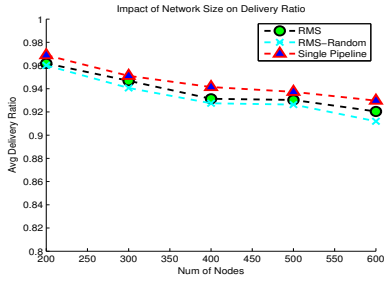
*2) **Impact of Link Quality**:* Previous work showed that wireless links are dynamic over time. Fig. 11 tests the effectiveness of RMS with different link quality settings. Fig. 11a shows Single Pipe is fragile and the delay could increase significantly when the link is unreliable, while RMS can improve the delay performance by resorting to multiple links. The gap between RMS and Single Pipeline becomes smaller with better links. This is because the transmission almost always succeed within the first several attempts. RMS would degrade to Single Pipeline if links are all perfect. As expected in Fig. 11b, energy cost increase with worse links due to multiple attempts in each one-hop delivery. Fig. 11c shows the delivery ratio can retain above 90% in a large range of links,



(a) avg delay vs. network size

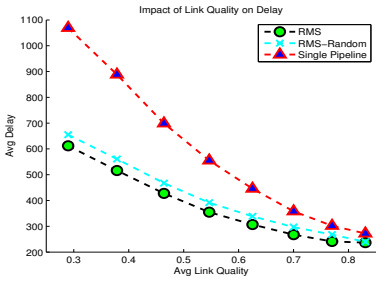


(b) num of transmissions vs. network size

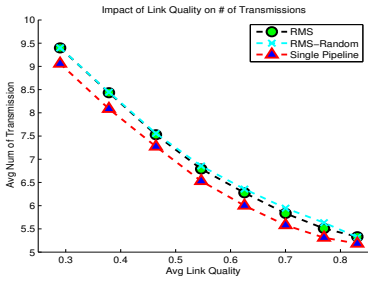


(c) delivery ratio vs. network size

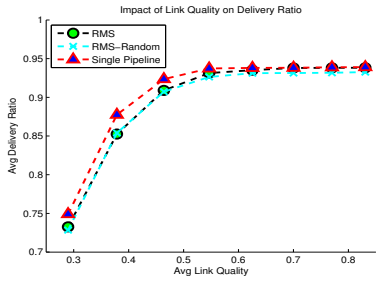
Fig. 10: Network Size



(a) avg delay vs. link quality

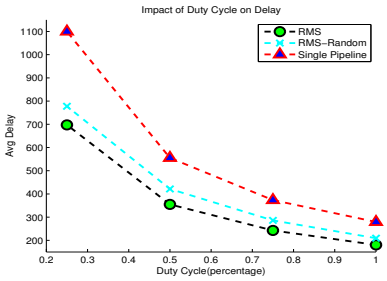


(b) num of transmissions vs. link quality

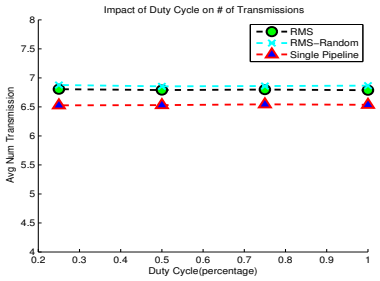


(c) delivery ratio vs. link quality

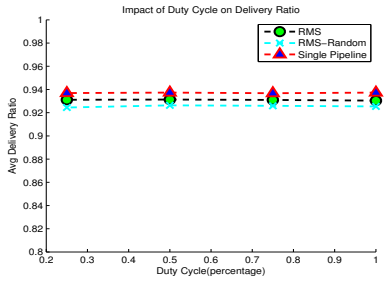
Fig. 11: Link Quality



(a) avg delay vs. duty cycle



(b) num of transmissions vs. duty cycle



(c) delivery ratio vs. duty cycle

Fig. 12: Duty Cycle

however, for those extremely bad links, delivery ratio could be improved further by increasing the system parameter R_{max} (recall R_{max} denotes the maximum number of retransmissions within one-hop delivery, as mentioned in section V-C).

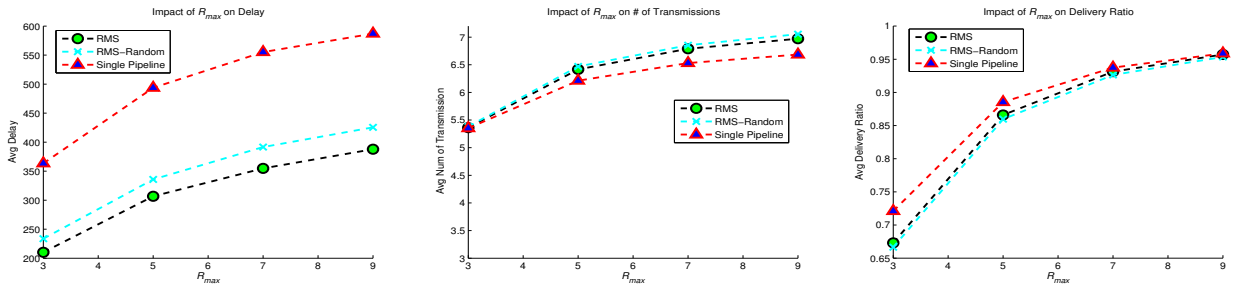
*3) **Impact of Duty Cycle**:* Fig. 12a proves the conclusion that the superiority of RMS over Single Pipeline design is more outstanding when the duty cycle is extremely low. For example, the gap between these two designs with duty cycle equaling 0.25% is much larger than that when duty cycle equals 1%. Fig. 12b and Fig. 12c shows the average energy cost and delivery ratio maintain almost the same for varying duty cycle because though duty cycle has impact on sleep latency, it will not change the expected number of transmission in each hop.

*4) **Impact of** R_{max}:* R_{max} denotes the maximum number of retransmissions allowed in each one-hop delivery. It is the system parameter which trades off energy cost and delivery ratio. Higher R_{max} indicates more generous tolerance on the failure of transmissions. Fig. 13a shows RMS outperforms Single Pipeline design by around 40% for all R_{max} settings. Higher R_{max} also increases the number of packets which are delivered successfully to the sink with large delay. Thus the average delay increases with larger R_{max}. Results in Fig. 13b and Fig. 13c reveal that greater R_{max} can be selected if high delivery ratio of the application is required though it could consume more energy.

VII. Implementation and Evaluation

A. Experiment Setup

Aside from large-scale simulations, we also implemented a prototype of RMS in our indoor TinyOS/MicaZ test-bed with 20 MicaZ nodes. As showed in Fig. 14, the motes are organized to engender a 4-hop network. Our experiment is composed of several phases including neighbor discovering and neighbor table setup, initial synchronization, link measurement, multi-pipeline scheduling, low-duty-cycle operation, packet delivery and final report to sink. Each node in the lowest level of the data gathering tree generates 10 packets



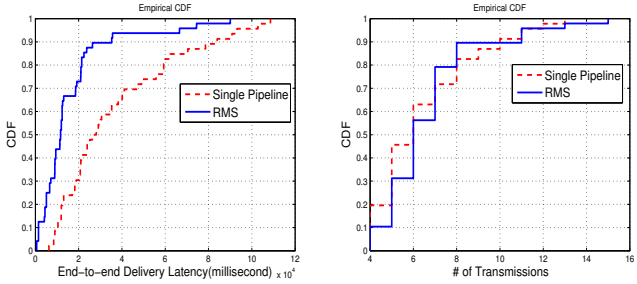
(a) avg delay vs. R_{max} (b) num of transmissions vs. R_{max} (c) delivery ratio vs. R_{max}

Fig. 13: Max Num of Retransmissions

Fig. 14: In-door Test-bed

and sends to the sink. We report the delivery latency for these packets since they actually indicate the end-to-end delay bound of the whole network.

B. Performance Comparison



(a) Delay (b) Transmissions

Fig. 15: Performance Comparison

Fig 15a reports the CDF of delivery latency for RMS and single pipeline design. In this experiment, each time slot is chosen as 60 milliseconds and the length of a period is $T = 200$, thus each working period takes up 12s. As Fig 15a shows, for single pipeline design, more than 40 percent of packets have the delivery latency beyond three working periods. This is due to broken pipeline and interrupted data forwarding. However, if RMS is applied, around 60 percent of packets can be delivered to the sink within one period because not only packets could be switched among different pipelines to handle a broken pipeline, but also these parallel pipelines are coordinated to reduce expected delivery latency. Fig 15b presents the number of transmissions required for these packets to be delivered to the sink. This figure verifies our conclusion that single pipeline is always fragile in reality since only 20 percent of packets can be delivered with four transmissions (the maximum hop count in our test-bed is four). As showed, although RMS could induce a little bit energy cost due to occasional attempts of transmissions on certain bad links, such overhead is almost tolerable especially considering the benefit gained in the performance of delivery latency.

VIII. Conclusion

Unreliable links would cause staggered wakeup scheduling or streamline scheduling design ineffective in data collection due to the fragility of pipeline. To overcome this realistic challenge and reduce the end-to-end delivery latency, we propose robust multi-parents scheduling (RMS) algorithm which combines the pipeline feature with the advantage of multi-parents forwarding. RMS can coordinate multiple pipelines and switch packet transmissions timely among different pipelines to reduce standby time. To evaluate the effectiveness of our design, we conduct large-scale simulations and in-door experiments showing that RMS can reduce the end-to-end delivery latency by around $40\% \sim 50\%$ within tolerable energy overhead.

Acknowledgment

This research was supported in part by the US National Science Foundation (NSF) grants CNS-0845994, CNS-0917097 and IBM OCR Fund.

References

[1] M.A.Batalin, M.Rahimi, Y.Yu, D. Liu, A. Kansal, G.S.Sukhatme, W. Kaiser, M. Hansen, G.J.Pottie, M. Srivastava, and D. Estrin, "Call and Response: Experiments in Sampling the Environment," in *SenSys'04*.

[2] Q. Zhang, G. Sobelman, and T. He, "Gradient-driven target acquisition in mobile wireless sensor networks," in *MSN'06*.

[3] Y. Zhang, L. Zhang, and X. Shan, "Robust Distributed Localization with Data Inference for Wireless Sensor Networks," in *ICC'08*.

[4] L. Mo, Y. He, Y. Liu, J. Zhao, S. Tang, X.-Y. Li, and G. Dai, "Canopy closure estimates with greenorbs: sustainable sensing in the forest." in *SenSys'09*, 2009.

[5] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay Efficient Sleep Scheduling in WIreless Sensor Networks," in *INFOCOM'05*.

[6] S. Guo, Y. Gu, B. Jiang, and T. He, "Opportunistic flooding in low-duty-cycle wireless sensor networks with unreliable links," in *MobiCom'09*.

[7] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency mac for data gathering," in *IPDPS '04*, 2004.

[8] Y. Li, W. Ye, and J. Heidemann, "Energy and latency control in low duty cycle MAC protocols," in *Proceedings of the IEEE Wireless Communications and Networking Conference*, 2005.

[9] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *IPSN '05*.

[10] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The Flooding Time Synchronization Protocol," in *SenSys'04*, 2004.

[11] R. Fonseca, O. Gnawali, K. Jamieson, and P. Levis, "Four bit wireless link estimation," in *HotNets VI*, 2007.

[12] M. Zuniga and B. Krishnamachari, "Analyzing the Transitional Region in Low Power Wireless Links," in *IEEE SECON'04*, 2004.