On Distributed Time-Dependent Shortest Paths over Duty-Cycled Wireless Sensor Networks

Shouwen Lai, Binoy Ravindran
Department of Electrical and Computer Engineering
Virginia Tech,Blacksburg, VA 24061
Email: {swlai,binoy}@vt.edu

***Abstract*—We revisit the shortest path problem in asynchronous duty-cycled wireless sensor networks, which exhibit time-dependent features. We model the time-varying link cost and distance from each node to the sink as periodic functions. We show that the time-cost function satisfies the FIFO property, which makes the time-dependent shortest path problem solvable in polynomial-time. Using the β-synchronizer, we propose a fast distributed algorithm to build all-to-one shortest paths with polynomial message complexity and time complexity. The algorithm determines the shortest paths for all discrete times with a single execution, in contrast with multiple executions needed by previous solutions. We further propose an efficient distributed algorithm for time-dependent shortest path maintenance. The proposed algorithm is loop-free with low message complexity and low space complexity of $O(maxdeg)$, where $maxdeg$ is the maximum degree for all nodes. The performance of our solution is evaluated under diverse network configurations. The results suggest that our algorithm is more efficient than previous solutions in terms of message complexity and space complexity.**

I. Introduction

The problem of routing over wireless sensor networks (WSNs) has attracted extensive attention in the recent years. Some WSN routing protocols [1], [2] presented in the literature are extended from routing protocols over wired/wireless ad-hoc networks. They find a path with the minimum hop count to the destination, which is based on the assumption that the link cost (or one-hop transmission delay) is relatively static for all links. However, for asynchronous duty-cycled WSNs, that assumption may not be valid.

In asynchronous duty-cycled WSNs, i.e., WSNs operating with MAC-layer protocols such as B-MAC [3] or X-MAC [4], sensor nodes operate in low power listening (LPL) mode. In the LPL mode, a node periodically switches between active and sleep state. The period for one active/sleep switching is called the LPL checking interval, which can be identical, or can be adaptively varied for different nodes, refereed to as ALPL [5]. The asynchronous duty-cycled mechanism have been shown to achieve excellent idle energy savings, scalability, and easiness in implementation. However, they suffer from time-varying neighbor discovery latencies which is also pointed out by Ye *et.al.* [6]. As shown in Figure 1, the neighbor discovery latency between two neighbors is varied with different departure times.

If we define the *link cost* as the time delay between data dispatching time at a sender and the data arrival time at the receiver, the link cost is time-varying in asynchronous duty-cycled WSNs due to varying neighbor discovery latency, even though the physical propagation condition does not change with time. This raises a non-trivial problem: with time-varying link costs, how to find optimal shortest paths with least nodes-to-sink latency for all nodes?

Previous works have modeled the similar problems as the time-dependent shortest path problem (or TDSP) [7], [8] in field of traffic networks [9], time-dependent graphs [10], and GPS navigation [11]. For distributed solutions for the problem, the only previous work [8] computes the shortest paths for a specific departure time in each execution. If the whole time period has M discrete intervals (M is ∞ for infinite time intervals), we have to execute the algorithm in [8] M times, which is inefficient in terms of message complexity and time complexity. Thus, our motivation is to design a fast distributed algorithm for the problem, which efficiently enumerates all the shortest paths for infinite time intervals.

Another motivation for our work is to dynamically and distributively maintain time-dependent shortest paths. In WSNs, a node may update its duty-cycle configuration (e.g., based on its residual energy), or join or leave the network, thereby changing the network topology. In such situations, the duty-cycle updating node or the joining/leaving node may change the cost of all the links with its neighbors, which means that a single node update can cause multiple link updates. Previous works on this problem [12], [13] are efficient in handling single link update. Applying such solutions for multiple link updates would imply that multiple distributed updates execute concurrently for a single node update, which is not efficient in terms of message and memory space complexities.

In this paper, we propose a distributed, time-dependent, shortest path routing protocol with low message and space complexities for duty-cycled WSNs. Our algorithm is based on the observation that the time-varying link cost function is periodic, and hence by derivation, the time-varying distance function for each node is also periodic. We show that the link cost function satisfies the FIFO property [7]. Therefore, the time-dependent shortest path problem is *not* an NP-hard problem, and therefore is solvable in polynomial time. We also propose distributed algorithms for maintaining the shortest paths. The proposed algorithms re-compute the routing paths based on previous path information. The message complexity of our algorithms is $O(\delta^2)$ per node update, where δ is the number of nodes that change either the distance or the parents in their shortest paths to the sink as a consequence of the corresponding nodes' update, and the space complexity is $O(maxdeg)$ which is scalable to large-scaled networks.

Our contributions are as follows: 1) We model asynchronous duty-cycled WSNs as time-dependent networks. We show that such networks satisfy the FIFO condition and the triangular path condition; and 2) We present distributed algorithms for finding the time-expanded shortest paths to the sink node for all nodes. When compared to the previous solution in [8], our algorithms find the shortest paths in a single execution for infinite time intervals; and 3) We present distributed shortest path maintenance algorithms with low message complexity and space efficiency.

To the best of our knowledge, we are not aware of any other efforts that consider duty-cycled WSNs as time-dependent networks and solve the problem of finding or updating the shortest paths with efficient message and space costs.

The rest of the paper is organized as follows: We outline our assumptions and define the problem in Section II. We formally model the link cost function and the distance function in Section III. The algorithms for route construction and route maintenance are explained in Section IV and V, respectively. We discuss some special cases for our proposed algorithms in Section VI. Simulations results are discussed in Section VIII. We survey past and related works in Section VII and conclude in Section IX.

II. Assumptions and Problem Definition

We model a WSN as a directed graph $G = (V, E, C)$, with $|V|$ nodes and $|E|$ links. $C = \{\tau_{i,j}(t) | (i,j) \in E\}$ is a set of time-dependent link delays, i.e., $\tau_{i,j}(t)$ is a strictly positive function of time defined for $[0, \infty)$, describing the delay of a message over link (i,j) at time t. Each node n_i only knows the identity of the nodes in its neighbor set, defined as N_i.

We assume that all nodes operate with duty-cycled LPL modes in the idle state. We define the duration of the LPL checking interval for node n_i as T_i. It is possible that $T_i \neq T_j$ (ALPL) for two nodes n_i and n_j. The time expansion of each node n_i is modeled as discrete and infinite, where $\mathfrak{T}_i = \{t_i^0, t_i^1, t_i^2, \cdots, t_i^M\}$, M is $+\infty$, and $t_i^k - t_i^{k-1} = T_i$. We use the terms of checking interval and time slot interchangeably.

We consider the policy of no waiting at each node since the node-to-sink delay will not benefit from waiting. Thus, once the data arrives at an intermediate node, the node will try to dispatch the data immediately. Dispatching times are not the same as the data departure times, as the data may still be buffered in the sender's memory. For simplicity in modeling and design, a node dispatches the received data at $t_i^k \in \mathfrak{T}_i$.

A nonnegative travel time $\tau_{i,j}(t_i^k)$ is associated with each link (i,j) with the following meaning: If t_i^k is the data dispatching time from node n_i along the link (i,j), then $t_i^k + \tau_{i,j}(t_i^k)$ is the data arrival time at node n_j.

The general problem of determining the shortest paths with the least delay in time-dependent WSNs can be defined as follows: Find the least time paths from all nodes to the sink node n_s corresponding to the minimum achievable delay d_i, $\forall n_i \in V$ and $\forall t_i^k \in \mathfrak{T}_i$, where

$$d_i(t_i^k) = \min_{n_j \in N_i} \{\tau_{i,j}(t_i^k) + d_j[t_i^k + \tau_{i,j}(t_i^k)]\} \tag{1}$$

Equation 1 is an extension of Bellman's equations [14] for the time-dependent network and is referred to as TD-Bellman's equation hereafter.

We assume that time slots in each node are numbered, and assume that the length of the data packets is fixed or relatively small, which is valid for a WSN designed for a specific application.

We also assume that a message arrives correctly with finite time from a sender to a receiver, which can be achieved by any reliable MAC-layer transmission mechanism. The network is not assumed to be time-synchronized, and nodes do not synchronize their wakeup/sleep schedules. We further assume that all nodes have the same time frequency, or their clocks drift at relatively slow speeds.

III. Modeling Asynchronous Duty-cycled WSNs

In this section, we model asynchronous duty-cycled WSNs as time-dependent networks. We show that the link cost function is periodic and establish that the time-varying distance function is also periodic. Having done so, we then implement TD-Bellman's Equation by vector operations.

A. Link Cost Function

Without loss of generality, suppose there are two adjacent nodes n_i and n_j, where n_i is the sender and n_j is the receiver.

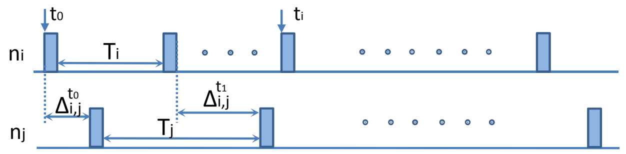


Fig. 1. Varying neighbor discovery latency in heterogenous LPL mode

Suppose at time t_i^0, the neighbor discovery latency is $\Delta_{i,j}^{t_i^0}$. Then at time $t_i^k = t_i^0 + k * T_i$, the neighbor discovery latency can be expressed as:

$$\Delta_{i,j}^{t_i^k} = T_j - (k * T_i - \Delta_{i,j}^{t_i^0}) mod\ T_j \tag{2}$$

In an actual deployment, we can measure $\Delta_{i,j}^{t_i^0}$ in the following way: at the beginning of the time slot which starts at t_i^0, node n_i sends out a preamble which contains the node ID of n_j, and n_j immediately feeds-back an ACK containing the value of T_j once it receives the preamble. After receiving the ACK by n_i, the one-hop round trip delay from t_i^0 to the time at which the ACK is received is set to $\Delta_{i,j}^{t_i^0}$. Once we measured $\Delta_{i,j}^{t_i^0}$, $\Delta_{i,j}^{t_i^k}$ $(k \geq 0)$ can be computed by Equation 2.

For data transmission with fixed length data packets, we define the data propagation time as τ_{data}. Now, for a directed link (i,j), we can set the link cost function as, for $\forall t_i^k \in \mathfrak{T}_i$,

$$\tau_{i,j}(t_i^k) = T_j - (k * T_i - \Delta_{i,j}^{t_i^0}) mod\ T_j + \tau_{data} \tag{3}$$

If τ_{data} is relatively small when compared with T_i and T_j, we can set $\tau_{data} = 0$. This is especially true for some WSN applications with small information reports, such as target tracking and environment monitoring.

Theorem 1: For every link (i, j), the time-varying link cost function is periodic. The minimum period for the function regarding k is,

$$P(\tau_{i,j}) = \frac{LCM(T_i, T_j)}{T_i} \tag{4}$$

where $\tau_{i,j}(t_i^k)$=$\tau_{i,j}(t_i^{k+P(\tau_{i,j})})$ $(k \geq 0)$ and LCM is the least common multiple.

B. Distance to Sink

We refer to the node-to-sink delay as *distance*, for compatible representation with that in the static Bellman-Ford algorithm [14].

Consider node n_i and its neighbor n_j, where $T_i \neq T_j$. For dispatching time t_i^0 at n_i, let us suppose that the data arriving time at n_j is $t_j^{j_0}$. Then, for the dispatching time t_i^k at n_i, with the same time frequency for the two nodes, the corresponding time instant at n_j is $t_j^{j_0} - \Delta_{i,j}^{t_i^0} + k * T_i$. Hence, based on the neighbor discovery mechanism (i.e.,B-MAC [3] and X-MAC [4]), n_j will be discovered by n_i at the time instant $t_j^{j_0} + \lceil \frac{k*T_i - \Delta_{i,j}^{t_i^0}}{T_j} \rceil * T_j$, with respect to n_j's time clock. Therefore, the function of distance of n_i to n_s from the path through n_j is:

$$d_i(t_i^k) = \tau_{i,j}(t_i^k) + d_j(t_j^{k'+j_0}) \tag{5}$$

for $\forall t_i^k \in \mathfrak{T}_i$, where $k' = \lceil (k * T_i - \Delta_{i,j}^{t_i^0})/T_j \rceil$ for $t_j^{k'} \in \mathfrak{T}_j$, and j_0 is the data arriving time slot at n_j with respect to dispatching time t_i^0.

Theorem 2: For a path $n_i \rightarrow n_{i-1} \cdots \rightarrow n_1 \rightarrow n_s$, the distance function $d_i(t_i^k)$,$\forall t_i^k \in \mathfrak{T}_i$, is a periodic function, where the minimum period for the function regarding k is:

$$P(d_i) = \frac{LCM(T_0, T_1, \cdots, T_i)}{T_i} \tag{6}$$

for $d_i(t_i^k) = d_i(t_i^{k+P(d_i)})$, where $T_0, T_1, \cdots, T_i$ are durations of the LPL checking intervals of nodes n_s, n_1, $\cdots$, n_i, respectively.

Proof: The proof is by induction. For $i = 1$, $d_1(t_1^k) = \tau_{1,s}(t_1^k) + d_s(t_1^{k'})$. Since $d_s \equiv 0$, $d_1(t) = \tau_{1,s}(t_1^k)$. According to Theorem 1, $d_1(t_1^k)$ is periodic and its period is $LCM(T_0, T_1)/T_1$. Assume the claim is true for node n_{i-1}.

Now for node n_i, $d_i(t_i^k) = \tau_{i,i-1}(t_i^k) + d_{i-1}(t_{i-1}^{k'+j_0})$, where $k' = \lceil (k * T_i - \Delta_{i,j}^{t_i^0})/T_j \rceil$ and j_0 is the data arrival time slot at n_{i-1} with respect to t_i^k, based on Equation 5. Let us define $f_1 = \tau_{i,i-1}(t_i^k)$ and $f_2 = d_{i-1}(t_{i-1}^{k'+j_0})$. The minimum period for function f_1 is $P(f_1) = LCM(T_i, T_{i-1})/T_i$. Let $\prod = LCM(T_0, T_1, \cdots, T_{i-1})$. Based on the induction step, for distance function $d_{i-1}(t_{i-1}^k)$, the period is $\prod/T_{i-1}$. The period for function f_2 is $P(f_2) = \prod/gcd(\prod, T_i)$. Because $gcd(\prod, T_i) = \frac{\prod * T_i}{LCM(\prod, T_i)}$, we have $P(f_2) = \frac{LCM(\prod, T_i)}{T_i}$.

Therefore, the minimum period for $d_i(t_i^k)$ is $P(d_i) = LCM[P(f_1), P(f_2)] = LCM(T_0, T_1, \cdots, T_i)/T_i$. ■

Given a WSN with different LPL checking intervals, the period for the distance function of any node is bounded by $LCM(T_0, T_1, \cdots, T_n)/\min\{T_i\}$, from Equation 6.

In practical implementations, it is recommended that $LCM(T_0, T_1, \cdots, T_n)/\min\{T_i\}$ is not arbitrarily large. Thus, our mechanism for finding the shortest paths at the routing layer should be based on cross-layer design. For example, {100ms, 200ms, 500ms, 1000ms} is a good configuration set, where there is a bounded period $LCM(100, 200, 500, 1000)/min\{100, 200, 500, 1000\} = 10$ for the distance function. It means that for any node, its distance to the sink will repeat at most every 10 checking intervals.

C. Implementation via Vectors

We implement the discrete, periodic, and infinite link cost functions and the distance functions as vectors, and implement the TD-Bellman's equation II by vector operations.

We implement the link cost function $\tau_{i,j}(t_i^k)$ $(t_i^k \in \mathfrak{T}_i)$ with a vector $\vec{\tau_{i,j}}$, where $|\vec{\tau_{i,j}}| = LCM(T_i, T_j)/T_i$ and $\tau_{i,j}[k]$ represents a set of numbers as follows:

$$\tau_{i,j}[k] = \{\tau_{i,j}(t_i^k), \tau_{i,j}(t_i^{k+|\vec{\tau_{i,j}}|}), \tau_{i,j}(t_i^{k+2*|\vec{\tau_{i,j}}|}), \cdots\} \tag{7}$$

For the node n_i, its distance function $d_i(t^k)$ $(t_i^k \in \mathfrak{T}_i)$ can be implemented by $\vec{d_i}$, where $|\vec{d_i}| = P(d_i(t^k))$. $d_i[k]$ represents a set of numbers as follows:

$$d_i[k] = \{d_i(t_i^k), d_i(t_i^{k+|\vec{d_i}|}), d_i(t_i^{k+2*|\vec{d_i}|}), \cdots\} \tag{8}$$

However, there are two difficulties for the implementation of the TD-Bellman's equation by vector operations.

The first one is vector mapping. To implement Equation 5, even if we know $\vec{\tau_{i,j}}$ and $\vec{d_j}$, we cannot add up the two vectors directly. We define a new vector $\vec{d_j'}$ by

$$d_j'[k] = d_j[(k' + j_0) \ mod \ |\vec{d_j}|] \tag{9}$$

where $k' = \lceil (k * T_i - \Delta_{i,j}^{t_i^0})/T_j \rceil$ and j_0 is the corresponding time slot for $\tau_{i,j}[0]$ at n_j (i.e., $t_i^0 + \Delta_{i,j}^{t_i^0} = t_j^{j_0}$).

Only after mapping $d_j[k]$ to $d_j'[k]$, we can add $\tau_{i,j}[k]$ to $d_j'[k]$. By vector mapping, the size for the new vector $\vec{d_j'}$ is:

$$|\vec{d_j'}| = \frac{|\vec{d_j}| * T_j}{gcd(|\vec{d_j}| * T_j, T_i)} \tag{10}$$

The second difficulty comes from the various sizes of vectors for link cost and distance. Suppose $d_i\vec{(j)}$ is the vector representing the distance of n_i from a path through n_j in discrete time intervals. To implement Equation 5, if $\vec{\tau_{i,j}}$ and $\vec{d_j'}$ have the same size, we can directly add them up for computing $d_i\vec{(j)}$. Otherwise, we need to expand the two vectors to be of the same size, which means expanding $\vec{\tau_{i,j}}$ by $LCM(|\vec{\tau_{i,j}}|, |\vec{d_j'}|)/|\vec{\tau_{i,j}}|$ times and $\vec{d_j'}$ by $LCM(|\vec{\tau_{i,j}}|, |\vec{d_j'}|)/|\vec{d_j'}|$ times. After the expansion, we can directly add up the expanded vectors. We call such an operation as *vector expansion*.

Vector mapping and expansion do not change the value of the discrete functions $\tau_{i,j}$ and d_j. They just change the representation of values of the two discrete functions. The vector expansion is valid since the time expansion is infinite.

We define the following functions for implementation:

Definition 1: For a vector $\vec{v}$:

- $ror(\vec{v}, ofs)$: output $\vec{v}'$ where $\forall k \in [0..|\vec{v}|-1]$, $v'[k] = v[(k+ofs) \ mod \ |\vec{v}|]$;
- $rol(\vec{v}, ofs)$: output $\vec{v}'$ where $\forall k \in [0..|\vec{v}|-1]$, $v'[k] = v[(|\vec{v}|+k-ofs) \ mod \ |\vec{v}|]$;
- $map(\vec{v}, a, b, \Delta, ofs)$: output v' where $\forall k \in [0..|\vec{v}|-1]$, $v'[k] = v[(\lceil \frac{a*k-\Delta}{b} \rceil + ofs) \ mod \ |\vec{v}|]$;
- $exp(\vec{v}, e) = \vec{v}||\vec{v}...||\vec{v}$ (e $times$) ($||$ presents catenating operation)

We utilize these functions to implement the TD-Bellman's equation. Suppose that n_i has received the distance vector $\vec{d_j}$ of node n_j. Suppose $\tau_{i,j}[0]$ is associated with the time slot $l_{i,j}^0$ at node n_i and the data arriving time slot for $\tau_{i,j}[0]$ is l_j^0 at n_j. Then, $\vec{d_i(j)}$, the distance vector of n_i to the sink from the path through n_j, can be calculated as:

$$\begin{aligned} \vec{d_j}' &= map[ror(\vec{d_j}, l_j^0), T_i, T_j, \tau_{i,j}[0], 0]; \\ \vec{d_i(j)}' &= exp(\vec{\tau_{i,j}}, e_1) + exp(\vec{d_j}', e_2); \\ \vec{d_i(j)} &= rol[\vec{d_i(j)}', l_{i,j}^0] \\ &= vec\_add(\vec{\tau_{i,j}}, \vec{d_j}, l_{i,j}^0, l_j^0) \end{aligned} \quad (11)$$

where $|\vec{d_j}'|$ is defined in Equation 10 and by defining $A = LCM(|\vec{\tau_{i,j}}|, |\vec{d_j}|)$, $e_1 = A/|\vec{\tau_{i,j}}|$ and $e_2 = A/|\vec{d_j}|$.

We update $\vec{d_i}$ and the corresponding parent vector $\vec{p_i}$ in the following way. Suppose the original $d_i[0]$ is associated with the time slot 0 at n_i (i.e., $d_i[0] = d_i(t_i^0)$). Now,

$$(\vec{d_i}, \vec{p_i}) = vmin\{exp(d_i, e_1^{'}), exp(\vec{d_i(j)}, , e_2^{'}]\} \quad (12)$$

where $B = LCM(|\vec{d_i(j)}|, |\vec{d_i}|)$, $e_1^{'} = B/|\vec{d_i}|$, and $e_2^{'} = B/|\vec{d_i(j)}|$. The function $(\vec{d_i}, \vec{p_i}) = vmin(\vec{v_1}, \vec{v_2})$ compares the corresponding elements in the two vectors $\vec{v_1}$ and $\vec{v_2}$, and copies the smaller element of each pair into the corresponding element in $\vec{d_i}$ and the corresponding vector ID into $\vec{p_i}$.

In addition, we define an operator $\vec{<}$ for comparing two vectors $\vec{v_1}$ and $\vec{v_2}$. Let $C = LCM(|\vec{v_1}|, |\vec{v_2}|)$. If $\forall k \in [0..C-1]$ $exp(\vec{v_1}, C/|\vec{v_1}|)[k] \leq exp(\vec{v_2}, C/|\vec{v_2}|)[k]$, then $\vec{v_1} \vec{<} \vec{v_2}$. For example, $[1,3] \vec{<} [2,3]$.

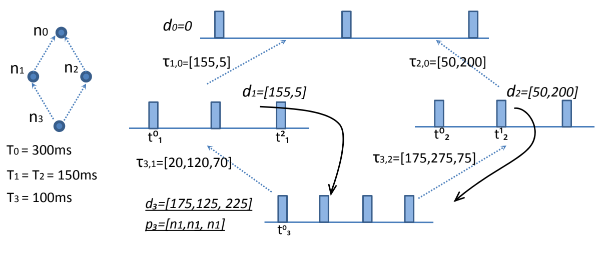


Fig. 2. Example for vector implementation

Now, we give an example to illustrate the vector implementation. In Figure 2, node n_1 sends its distance vector d_1 to n_3 by a message containing d_1, and node n_2 sends its distance vector d_2 to n_3 by a message containing d_2, where $d_1 = [155, 5]$ and $d_2 = [200, 50]$. Suppose $\tau_{1,0}$, $\tau_{2,0}$,$\tau_{3,1}$ and $\tau_{3,2}$ are measured in their 0^{th} slot respectively.

After n_3 receives the message from n_1, it computes the distance vector $\vec{d_3(1)}$ based on Equation 11. We have $\vec{d_1'} = [155, 5, 155]$ and $\vec{d_3(1)} = [20, 120, 70] + [155, 5, 155] = [175, 125, 225]$. A similar computation will happen when receiving the message (d_2,1) and we can obtain $\vec{d_3(2)} = [375, 325, 275]$. Then, $vmin\{[175, 125, 225], [375, 325, 275]\}$ will return $([175, 125, 225], [n_1 \ n_1 \ n_1])$ containing the shortest distance and the corresponding parents for n_3.

D. Properties

Theorem 3: FIFO condition: The link cost function $\tau_{i,j}(t_i^k)$ satisfies the FIFO property, which means, for any $t_i^{k_1} < t_i^{k_2}$,

$$t_i^{k_1} + \tau_{i,j}(t_i^{k_1}) \leq t_i^{k_2} + \tau_{i,j}(t_i^{k_2}) \quad (13)$$

Proof: From Equation 3, we have $t_i^{k_2} + \tau_{i,j}(t_i^{k_2}) - t_i^{k_1} - \tau_{i,j}(t_i^{k_1}) = (k_2 - k_1) * T_i + (k_1 * T_i - \Delta_{i,j}^{t_i^0}) \ mod \ T_j - (k_2 * T_i - \Delta_{i,j}^{t_i^0}) \ mod \ T_j = (k_2 - k_1) * T_i + [(k_1 - k_2) * T_i] \ mod \ T_j = (k_2 - k_1) * T_i - [(k_2 - k_1) * T_i] \ mod \ T_j \geq 0$. ■

By Theorem 3, the time-dependent shortest path problem in asynchronous duty-cycled WSNs is not NP-hard and is solvable in polynomial-time [7].

Theorem 4: Suppose node n_i has two neighbor n_j and n_k which are one-hop away from each other. Then, at a time instant, $t_i^{l_i}$, we have the triangular property:

$$\tau_{i,j}(t_i^{l_i}) \leq \tau_{i,k}(t_i^{l_i}) + \tau_{k,j}(t_i^{l_i} + \tau_{i,k}(t_i^{l_i})) \quad (14)$$

Proof: Suppose at time $t_i^{l_i}$, the data arriving time slot at n_j is $t_j^{l_j}$, and the data arriving time at n_k is $t_k^{l_k}$.

If $t_k^{l_k} \leq t_j^{l_j}$, which means that the data arriving time at n_k is earlier than the data arriving time at n_j, $\tau_{i,k}(t_i^{l_i}) + \tau_{k,j}(t_i^{l_i} + \tau_{i,k}(t_i^{l_i})) = t_k^{l_k} - t_i^{l_i} + t_j^{l_j} - t_k^{l_k} = t_j^{l_j} - t_i^{l_i} = \tau_{i,j}(t_i^{l_i})$.

If $t_k^{l_k} > t_j^{l_j}$, which means that the data arriving time at n_k is later than the data arriving time at n_j, we have $\tau_{i,k}(t_i^{l_i}) + \tau_{k,j}(t_i^{l_i} + \tau_{i,k}(t_i^{l_i})) = t_k^{l_k} - t_i^{l_i} + t_j^{l_j'} - t_k^{l_k} > t_j^{l_j} - t_i^{l_i} + t_j^{l_j'} - t_k^{l_k} > t_j^{l_j} - t_i^{l_i} > \tau_{i,j}(t_i^{l_i})$. The theorem follows. ■

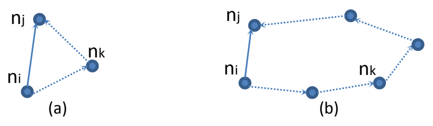


Fig. 3. Triangular path condition

Theorem 4, as illustrated in Figure 3(a), illustrates that node n_i will always arrive at its neighbor n_j directly without through other nodes. We have the following claim.

Lemma 1: Triangular Path Condition: For a node n_i and its neighbor n_j, at any dispatching time, the one-hop path $n_i \rightarrow n_j$ always has the least time delay.

We omit the proof for the triangular path condition since it is a simple extension from that of Theorem 4. An illustration is given in Figure 3(b). Note that the triangular path condition does not exist in static networks.

IV. Initial Route Construction

We now present distributed algorithms for initial time-dependent shortest path route construction in asynchronous duty-cycled WSNs, where the distances from all nodes to the sink node are initially infinite.

The proposed algorithm referred to as the FTSP algorithm, Fast Time-Dependent Shortest Path algorithm, is inspired by existing algorithms [8], and is adapted from the distributed Bellman-Ford algorithm augmented with β-synchronizer [15]. Comparing with the solution in [8], which only computes the shortest paths for a given specified discrete time, FTSP compute the shortest paths for infinite discrete time intervals in one execution.

Let $|D_m|$ denote the diameter of the longest shortest path for all nodes. We show that the message complexity of FTSP is $O(|D_m||E|)$ and the time complexity is $O(|D_m||V|)$. FTSP does not suffer from exponential message complexity, like in previous work [16] for the static shortest path problem over asynchronous networks.

A. Distributed Algorithm Description

We present the data structures and message formats in FTSP:

- $\vec{d_i}$: vector of distance from n_i to n_s, defined in Equation 8; initially all elements are ∞;
- $\vec{\tau_{i,j}}$: link delay from n_i to its neighbor n_j, defined in Equation 7; $\tau_{i,j}[0]$ is obtained by measurement;
- $\vec{p_i}$: vector of parents for n_i in the shortest path $n_i \rightarrow n_s$ for infinite time intervals; initially all elements are node n_i;
- $MSG(ID_{src}, ID_{from}, \vec{d}, updated)$: control message; ID_{src} is the node ID of sink node, or update source node (see Section V); ID_{from} is the sender's node ID; $\vec{d}$ is the distance vector of the sender; $updated$ is to show whether there is update in the current iteration, which will be explained later;
- $ACK[j]$: boolean indicating whether a node receives a control message from its neighbor n_j

We assume that n_i knows the duration of the checking interval T_j of all its neighbors $n_j \in N_i$ after measuring link delays. Initially, a directed spanning tree rooted at n_s is built upon the network. We assume that n_i knows its parent st_p_i in the spanning tree. We also assume that FTSP is invoked by higher-level protocols that create "START" impetuses at n_s.

Algorithm 1: Algorithm for sink node n_s in FTSP:

```
Initialization:
  $\forall n_j \in N_s$, $ACK[j] = 0$, $updated[j] = true$;
On receiving START:
  send MSG(s, s, 0, false) to $\forall n_j \in N_s$;
On receiving MSG(j,$\vec{d_j}$, $l_j$, bChanged):
  $ACK[j] = 1$;   $updated[j] = bChanged$;
  if ($\forall n_j \in N_s, ACK[j] == 1$) then
      $ACK[j] = 0$;
      if ($\forall n_j \in N_s$, $updated[j] == false$) then
          STOP;
    else
          send MSG(s, s, 0, false) to $\forall n_j \in N_s$;
```

The first iteration of FTSP begins when node n_s receives the "START" impetus. Subsequent ones begin whenever n_s completes an iteration and determines whether another one is necessary by checking whether there is a node whose distance was minimized in the last iteration.

Each iteration begins at node n_s by sending a control message to all its neighbors. When replies from all its neighbors have been received, node n_s concludes that an iteration is completed. Every other node, i.e., n_i ($n_i \neq n_s$), begins an iteration upon receiving a control message from its parent st_p_i in the spanning tree, upon which it sends control messages to all its neighbors except st_p_i. When replies are received from all these neighbors, a control message is sent to the parent, thereby completing the current iteration at node n_i.

The control message from node n_j contains the distance vector $\vec{d_j}$ (known thus far during the previous iterations) between n_j and n_s. When such a message is received at node n_i, node n_i checks whether the new information minish the value of any element in the current distance vector. It does so by considering the path that goes through n_j, taking into account the most recent information from n_j.

We describe the procedures in Algorithms 1 and 2.

Algorithm 2: Algorithm for node $n_i (i \neq s)$ in FTSP:

```
Initialization:
  $st\_p_i = NULL$;
  for $\forall n_j \in N_i$ do
      $ACK[j] = 0$; $updated[j] = true$;
      Measure link delay $\Delta^0_{i,j}$ at the beginning of any time slot;
      $l^0_{i,j}$ = the time slot number for measurement;
      $l_j$ = the data arriving time slot number at $n_j$;
      for $\forall k \in [0..LCM(T_i, T_j)/T_j]$ do
          $\tau_{i,j}[k] = T_j - (k * T_i - \Delta^0_{i,j}) mod\ T_j$;

On receiving MSG($s$, $j$, $\vec{d_j}$, bChanged) from $n_j$:
  $ACK[j] = 1$;  updated[j] = bChanged;  $\vec{d_i}^{prev} = \vec{d_i}$;
  if $n_j == st\_p_i$ then
      send MSG(s, $i$, $\vec{d_i}$, false) to $\forall n_j \in N_i$ except $st\_p_i$;

  $\vec{d_i(j)}$ = vec_add($\vec{d_i}, \vec{d_j}, l^0_{i,j}, l_j$); /* Equation 11*/
  ($\vec{d_i}, \vec{p_i}$) = vmin($\vec{d_i}, \vec{d_i(j)}$);      /* Equation 12*/
  if ($\vec{d_i} \vec{<} \vec{d_i^{prev}}$) then updated[j] = true;
  if ($\forall n_j \in N_i, ACK[j] == 1$) then
      if ($\exists n_j \in N_i, updated[j] == true$) then bUpdated = true;
      else then bUpdated = false;
      send MSG($s$, $i$, $\vec{d_i}$, bUpdated) to $st\_p_i$;
      $\forall n_j \in N_i, ACK[j] = 0$;
```

The functions $vec\_add(\cdot)$, $vmin(\cdot)$, and operator $\vec{<}$ are defined in Section III-C.

B. Correctness and Complexity

Let $PATH(i, t_i^k)$ be a path obtained by node n_i, which is starting at time t_i^k and moving along its parent $p_i[k]$. Let $d_i[k]_m$ denote the value of $d_i[k]$ after the m^{th} iteration in FTSP. We have the following properties:

Theorem 5: 1) After termination, $PATH(i, t_i^k)$ is loop-free and concatenated; and 2) In the m^{th} ($m \geq 0$) iteration, a node n_i whose shortest path is at most m-hop away from the sink node for all discrete time intervals $t_i^k \in \mathfrak{T}_i$ will be determined.

Proof: For part 1, $\forall t_i^k \in \mathfrak{T}_i$, after termination, suppose $p_i[k]$ is set to the node n_j for the shortest path with respect to n_s. Since n_j is the parent of n_i at the time slot t_i^k, $PATH(i, t_i^k)$ is a path composed of $PATH(j, t_i^k + \tau_{i,j}(t_i^k))$ which is appended to node n_i $\forall t_i^k \in \mathfrak{T}_i$. Thus, for any node n_i and $\forall t_i^k \in \mathfrak{T}_i$, $PATH(i, t_i^k)$ is concatenated.

We prove that $PATH(i, t_i^k)$ is loop-free by contradiction. Without loss of generality, assume that there is a loop and the loop is $(n_{i_0} \rightarrow n_{i_1} \rightarrow n_{i_2} \cdots n_{i+k} \rightarrow n_{i_0})$. This means that there is a shortest path $(n_{i_0} \rightarrow n_{i_1} \cdots, \rightarrow, n_{i+k})$, where n_{i+k} is one-hop away from n_{i_0}. According to the triangular path condition in Lemma 1, such a shortest path cannot exist because $n_{i_0} \rightarrow n_{i+k}$ is always the shortest one among all paths from n_{i_0} to n_{i+k}, contradicting the assumption.

We prove part 2 by induction on m. It is easy to find that the claim is true for $m = 0$. Now, assume that the claim is true for $m - 1$ (i.e., the inductive hypothesis). We prove for m by induction.

Consider a specific time t_i^k and a node n_i such that there is a shortest path with at most m hops between n_i and n_s. Let $SP(i, s, t_i^k, m)$ be the shortest path which is at most m hops from n_i to n_s at time t_i^k. Let n_j be n_i's parent on $SP(i, s, t_i^k, m)$ at t_i^k. This means that there is a path with at most $m - 1$ hops between n_j and n_s.

By the inductive hypothesis, n_j determined its shortest distance at the $(m-1)^{th}$ iteration. In the m-*th* iteration, node n_j sends its minimized distance vector $\vec{d}_{j_{m-1}}$ to node n_i. Thus, $SP(i, s, t_i^k, m)$ is determined after receiving the vector $\vec{d}_{j_{m-1}}$ in the m^{th} iteration, which completes the inductive step. Since t_i^k is chosen arbitrarily, this holds for all values of $t_i^k \in \mathfrak{T}_i$. ■

The part 2 in Theorem 5 implies that, for $m \geq D_m$, all nodes determine their minimum delay and the corresponding parents for all time intervals, since all shortest paths contain at most D_m nodes.

Theorem 6: The message and time complexity of FTSP is $O(D_m|E|)$ and $O(D_m|V|)$ respectively.

Proof: Based on the implementation of the β-synchronizer in [17], in each iteration, there is exacted one message traversing each link in the spanning tree, totally $|E|$ messages exchanged. By Theorem 5, the number of iterations is upper bounded by the longest shortest path's length D_m. Thus, the message complexity is $O(D_m|E|)$.

Suppose the largest delay for transmitting a message in the spanning tree is a constant, denoted by $|C|$. In each iteration, the time consumed is at most $|V| * |C|$. Since there are at most D_m iterations, the time complexity is $O(D_m|V|)$. ■

V. Dynamic Route Maintenance

When compared with static networks, link changes and node changes are more frequent in duty-cycled WSNs. If a node changes its duty-cycle configuration, or dynamically joins or leaves the network, the links connecting with all its neighbors will be changed at multiple time intervals. In such a situation, a single node update usually causes multiple link updates.

Some previous works (e.g., [12]) in static networks have proposed solutions that efficiently deal with single link updates. They are inefficient for multiple link updates caused by a single node update. The algorithms in [12] are also memory-inefficient, since each node stores the route entries for all other nodes, incurring the space complexity of $O(|V|)$.

The proposed algorithms, referred as to FTSP-M ("M" meaning maintenance), focus on per node update and can be easily extended to node insertion and deletion. If there are multiple node updates, the algorithms will run concurrently at multiple nodes.

A. Distributed Algorithm Descriptions

The proposed distributed algorithms for path maintenance are also equipped with $\beta-$synchronizer. We use similar data structures and message formats as that in Section IV-A.

Suppose the source update node is n_u and the corresponding input change is σ. We divide σ into two parts: σ_{inc} and σ_{dec}, where σ_{inc} includes the increasing links for $\forall t_u^k \in \mathfrak{T}_u$, and σ_{dec} includes the decreasing links for $\forall t_u^k \in \mathfrak{T}_u$. Let $\delta(\sigma_{inc})$ be the set of nodes that change either the distance or the parents for all infinite discrete time intervals, as a consequence of σ_{inc}. Similarly, let $\delta(\sigma_{dec})$ be the set of nodes affected by σ_{dec}. Apparently, $\delta(\sigma) = \delta(\sigma_{inc}) \cup \delta(\sigma_{dec})$.

FTSP-M consist of two phases for node n_u and all nodes $n_i \in \delta(\sigma)$ as described in Algorithms 3 and 5.

In phase 1, an initial spanning tree is built up gradually to contain all nodes in $\delta(\sigma_{inc})$. The purpose of phase 1 is to let all nodes in $\delta(\sigma_{inc})$ increase their distances to the sink node as a consequence of σ_{inc}, along the time-expanded shortest path trees rooted at n_u. After the termination of phase 1, all nodes in $\delta(\sigma)$ will never increase their distance again.

The rationale here is to satisfy the sufficient loop-free condition claimed in [18]: If at time t, node n_i detects a link-cost decrease or a decrease in the distance reported by a neighbor, then node n_i is free to choose its new parents. This is referred to as the *distance increase condition* (or DIC).

In each iteration, node n_u sends a control message to all neighbors. Every other node, i.e., n_i $(n_i \neq n_u)$ will send control messages to all its neighbors if it is in the spanning tree and receives a message from its parent. If n_i is not in the spanning tree in the current iteration, it checks whether n_j is its parent in its shortest path after receiving a control message from n_j, which can be done by checking whether $n_j \in \vec{p_i}$. If true, n_v will join the spanning tree and set $newsp_i = n_j$. By doing so, the spanning tree will increase at most one level in each iteration.

A control messages will traverse from the root (n_u) to all other nodes in the spanning tree just like that in FTSP. When node n_i receives a control message from n_j, it only updates the element to be increased in its distance vector, as illustrated in Algorithm 4.

When replies from all its neighbors have been received, node n_u concludes that an iteration is completed. When replies are received from all neighbors by a node, a control message is sent to the parent, thereby completing the iteration at the node. When there is no distance increase for all nodes in $\delta(\sigma_{inc})$, phase 1 will be terminated and node n_u will start phase 2.

Algorithm 3: Operations at update node n_u in FTSP-M:

```
Initialization:
  Same initialization as in Algorithm 2;
  if $p_u \neq null$ then
      for $\forall n_j \in N_u$ do
          $(\vec{d_j}, l_j^0)$ = get_dist($n_j$); /* retrieve $\vec{d_j}$, detailed
          implementation is omitted */
          $\vec{d_j}(u)$=vec_add($\vec{\tau_{u,j}}, \vec{d_j}, l_{u,j}^0, l_j^0$); /* Equation 11*/
          inc_update($\vec{d_u}, \vec{p_u}, n_j, \vec{d_j}(u)$);
Phase 1: On receiving START:
  send MSG($u$,$u$,$\vec{d_u}$,false) to $\forall n_j \in N_u$;
Phase 1: On receiving MSG($u$, $j$, $\vec{d_j}$, bChanged):
  $ACK[j] = 1$;   $updated[j] = bChanged$;
  if ($\forall n_j \in N_u, ACK[j] == 1$) then
      if ($\forall n_j \in N_u$, $updated[j] == false$) then
          Beginning Phase 2;
      else
          send MSG($u$,$u$,$\vec{d_u}$,false) to $\forall n_j \in N_u$;
          $\forall n_j \in N_u, ACK[j] = 0$;
Phase 2: On Beginning Phase 2:
  send MSG($u$,$u$,$\vec{d_u}$,false) to $\forall n_j \in N_u$;
Phase 2: On receiving MSG(0, $j$, $\vec{d_j}$, bChanged) from $n_j$:
  $ACK[j] = 1$;   $updated[j] = bChanged$;   $\vec{d_u^{prev}} = \vec{d_u}$;
  $\vec{d_u(j)}$ = vec_add($\vec{d_u}, \vec{d_j}, l_{i,j}^0, l_j,$); /* Equation 11 */
  $(\vec{d_u}, \vec{p_u})$ = vmin($\vec{d_u}, \vec{d_i(j)}$);      /* Equation 12 */
  if ($\vec{d_u} \lessgtr \vec{d_u^{prev}}$) updated[j] = true;
  if ($\forall n_j \in N_u, ACK[j] == 1$) then
      if ($\forall n_j \in N_u$, $updated[j] == false$) then
          STOP;
      else
          send MSG($u$,$u$,$\vec{d_u}$,false) to $\forall n_j \in N_u$;
          $\forall n_j \in N_u, ACK[j] = 0$;
```

Algorithm 4: Function: inc_update($\vec{d_1}$,$\vec{p_1}$,n_2,$\vec{d_2}$)

```
$d_1$ = exp($\vec{d_1}$,$(|\vec{d_1}| * |\vec{d_2}|)/|\vec{d_1}|$); $d_2$ = exp($\vec{d_2}$,$(|\vec{d_1}| * |\vec{d_2}|)/|\vec{d_2}|$);
$p_1$ = exp($\vec{p_1}$,$(|\vec{d_1}| * |\vec{d_2}|)/|\vec{d_1}|$); flag = false;
for $k = 0$ to $|\vec{d_1}| * |\vec{d_2}| - 1$ do
    if $p_1[k] == n_2$ && $d_1[k] < d_2[k]$ then
        $d_1[k] = d_2[k]$; flag = true;
return flag;
```

In phase 2, the initial spanning tree built up in *phase 1* is continuously growing until it contains all nodes in $\delta(\sigma)$. Phase 2 is also running by iterations. In each iteration, when a node n_i not in the spanning tree receives a control message from n_j, if the value of any elements in its distance vector can be minished, it will join the spanning tree by setting its parent $newsp_i$ to n_j. The distance update and message traversing in phase 2 of FTSP-M are just similar to that in FTSP.

B. Correctness and Complexity

In phase 1, all nodes in $\delta(\sigma_{inc})$ do not change their parents, but increase their distances as a consequence of σ_{inc}. Thus, there is no loop in phase 1. In phase 2, all nodes in $\delta(\sigma)$ will never increase their distances, thereby satisfying the distance increase condition [18]. All paths are therefore loop-free.

Theorem 7: In phase 1, each node in $\delta(\sigma_{inc})$ with at most m hops away from n_u along the time-dependent shortest path will not increase its distance after m iterations.

Algorithm 5: Operations in node $n_i (n_i \neq n_u)$ in FTSP-M:

```
Initialization: $newsp_i = null$; $\forall n_j \in N_i$,$updated[j] = false$;
Phase 1: On receiving MSG($u$, $j$, $\vec{d_j}$, bChanged) from $n_j$:
  if $n_j \in N_u$ then
      re-measure $\tau_{i,u}$ at the beginning of one time slot;
      reset $l_{i,j}^0$ and $l_u^0$;
  $\vec{d_i}(j)$=vec_add($\vec{\tau_{i,j}}, \vec{d_j}, l_{i,j}^0, l_j^0$);
  if $newsp_i == null$ then
      if (inc_update($\vec{d_i}, \vec{p_i}, n_j, \vec{d_i}(j)$)) then
          $newsp_i = n_j$;  send MSG($u$,$i$,$\vec{d_j}$,true) to $n_j$;
      else send MSG($u$, $i$, $\vec{d_j}$, false) to $n_j$;
  else
      FORWARD(u); /* $u$ means the MACRO is executed in phase 1 */
      $ACK[j] = 1$;   updated[j] = inc_update($\vec{d_i}, \vec{p_i}, n_j, \vec{d_i}(j)$);
      ACK_REPLY(u);
Phase 2: On receiving MSG(0, $j$, $\vec{d_j}$, bChanged):
  if $newsp_i == null$ then
      if ($updated[j]$) then
          $newsp_i = n_j$;   send MSG(0, $i$, $\vec{d_j}$, true) to $n_j$;
      else send MSG(0, $i$, $\vec{d_j}$, false) to $n_j$;
  else
      FORWARD(0); /* 0 means the MACRO is executed in phase 2 */
      $ACK[j] = 1$;   $updated[j] = bChanged$;   $\vec{d_i^{prev}} = \vec{d_i}$;
      $\vec{d_u(j)}$ = vec_add($\vec{d_u}, \vec{d_j}, l_{i,j}^0, l_j$); /* Equation 11*/
      $(\vec{d_u}, \vec{p_u})$ = vmin($\vec{d_u}, \vec{d_i(j)}$);      /* Equation 12*/
      if ($\vec{d_u} \lessgtr \vec{d_u^{prev}}$) updated[j] = true;
      ACK_REPLY(0);
FORWARD(int dict): code macro
  if ($newsp_i == n_j$) then
      send MSG(dict,$i$,$\vec{d_i}$,false) to $\forall n_j \in N_i$ except $newsp_i$;
ACK_REPLY(int dict): code macro
  if ($\forall n_j \in N_i, ACK[j] == 1$) then
      if ($\exists n_j \in N_i, updated[j] == true$) then bUpdated = true;
      else then bUpdated = false;
      send MSG(dict, i, $\vec{d_i}$,bUpdated) to $newsp_i$;
      $\forall n_j \in N_i, ACK[j] = 1$; bUpdated = false;
```

We omit the detailed proof due to space limitations. It can be done by induction similar to that for part 2 in Theorem 5. By Theorem 7, after $|\delta(\sigma_{inc})|$ iterations, all nodes in $\delta(\sigma_{inc})$ will not increase their distance anymore.

Definition 2: updated-subpath: for any node $n_i \in \delta$, the updated-subpath is from n_i to the first node n_e not in δ along the shortest path from n_i to n_s.

Theorem 8: In phase 2, all generated updated-subpaths are loop free, and updated-subpaths with at most m hops long are determined in the m^{th} iteration.

Proof: After phase 1, the DIC loop-free condition [18] is satisfied. Thus, all updated-subpaths are loop free in phase 2. The proof for at most m hops-long updated-subpaths being determined in the m^{th} iteration can be done by induction as that for the part 2 in Theorem 5. We omit the detailed proof here due to space limitations. ■

Theorem 9: The message complexity for per node update with $\delta(\sigma)$ output change is $O(|\delta(\sigma)|^2 * maxdeg)$. The time complexity is $O(|\delta(\sigma)|^2)$, and space complexity is $O(maxdeg)$.

Proof: In phase 1, the number of iterations is $\delta(\sigma_{inc}) \leq \delta(\sigma)$. In phase 2, there are at most $\delta(\sigma)$ iterations before all updated-subpaths are decided. Thus totally, there are $O(|\delta(\sigma)|^2 * maxdeg)$ messages. In each iteration, the consumed time is at most $|\delta(\sigma)| * C$ (C is the largest transmission delay for all links). Thus, the message complexity is $O(|\delta(\sigma)|^2)$. Since a node only stores the information of all its neighbors, the space complexity is $O(maxdeg)$. ■

VI. Discussion

FTSP in Section IV is a proactive routing protocol. Although its time complexity is $O(D_m * |V|)$ for initial route construction, it is affordable in the initial stage of WSNs. The low space complexity ($O(|maxdeg|)$) for route maintenance makes the algorithm scalable for large-scale WSNs.

Note that FTSP and FTSP-M target the ALPL mode [5] with various checking intervals. When all nodes have homogenous LPL checking intervals (like that in standard B-MAC), according to Equations 3 and 5, the link cost function and the distance function will become constants. In such case, our algorithm will default to the static shortest path algorithm. However, FTSP and FTSP-M will yield the same message complexities and time complexities for the static situation.

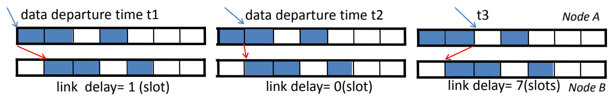


Fig. 4. Varying neighbor discovery delay for quorum wakeup scheduling

In addition, FTSP and FTSP-M can be applied for asynchronous quorum-based wakeup scheduling [19] in wireless sensor/ad-hoc networks. In asynchronous wakeup scheduling, the neighbor discovery delay may vary at different time moments as illustrated in Figure 4. The time-varying link cost function and the distance function are also periodic, and FTSP and FTSP-M are directly applicable.

VII. Past and Related Works

We summarize the related literature on LPL scheduling and time-dependent shortest path problem as follows.

LPL/ALPL in WSNs: B-MAC [3] is a widely used asynchronous LPL MAC protocol in WSNs. B-MAC provides interfaces that support configurable LPL parameters. X-MAC [4] is an improved version of B-MAC. Jurdak *et. al.* [5] and Vigorito *et. al.* [20] present adaptive low power listening (ALPL) mode based on node residual energy. These works provide the application spaces for our work.

Time-Dependent Shortest Path Problem: This problem was first proposed by Cooke and Halsey [7]. It has been well studied in the field of traffic networks [9], time-dependent graphs [10], and GPS navigation [11]. For the distributed time-dependent shortest path problem, the only previous work [8] computes the shortest paths for a specific departure time in each execution, which is not time-efficient.

Dynamic Shortest-Path maintenance: Many works [13], [12], [21] exist for handling link decreases and increases, and node deletions and insertions in static networks. These algorithms [12], [21] need $O(n)$ space at each node, which is impractical for sensor nodes with limited memory capacity. In addition, none of the previous works are efficient for shortest path maintenance in time-dependent networks.

VIII. Simulation Results

We evaluated the performance of FTSP and FTSP-M through extensive simulations using the OMNET++ simulator [22]. We compared our algorithms against other related algorithms for the TDSP problem, including the distributed Bellman-Ford algorithm [16] adapted to the time-dependent model (Section III), referred to here as TD-Bellman, and DSPP1 in [8]. The following three major metrics were measured in the evaluation: 1) message count; 2) time cost, which is the total time slots needed for stabilization; and 3) average memory cost. We examined two main factors that affect the performance of our protocols, including network size and time slot lengths. Our experimental settings were compatible with typical configurations in [6], [5], [2]. The wireless communication range was set to 10m. We adopt the wireless loss model used in [23], which considers the oscillation of radio links. The size of the data packets was fixed as 512 bytes.

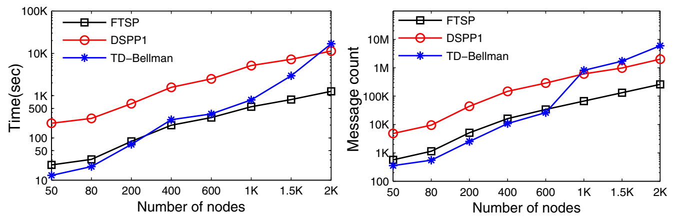
We generated 8 network size sets with varying sizes, $G_1,\cdots,G_8$, which are listed in Table I. For each network size, we randomly generated 10 topologies. Each data point presented in this section is the average of 10 topologies with 10 runs on each topology.

TABLE I
Network Size Sets

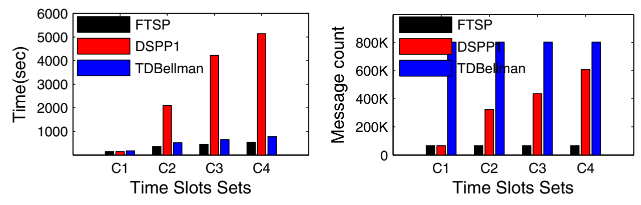
	G_1	G_2	G_3	G_4
$\|V\|$	50	80	200	400
	G_5	G_6	G_7	G_8
$\|V\|$	600	1K	1.5K	2K

TABLE II
Time slots sets

C_1 (ms)	$\{100, 100, 100, 100\}$
C_2 (ms)	$\{100, 200, 300, 600\}$
C_3 (ms)	$\{100, 200, 400, 800\}$
C_4 (ms)	$\{100, 200, 500, 1000\}$



(a) Varying $|V|$



(b) Varying time slots sets

Fig. 5. Comparison of time efficiency and message efficiency

We also varied the durations of time slots in the simulation with 4 sets, C_1, C_2, C_3, and C_4, as listed in Table II. With each set, we randomly choose one element as the value of LPL checking interval for each node.

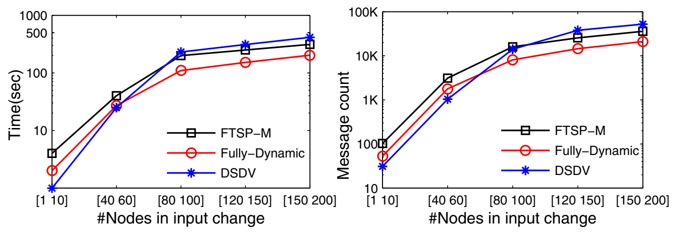
A. Route Construction

In the first set of simulations, we fixed the time slots set by C_4, which indicates that the largest distance vector size is 10 by Equation 6, and varied the network size. With the number of nodes increasing from 50 to 2000 in $G_1,\cdots,G_8$, the average time consumed and the message count are shown in Figure 5(a). The average execution time of DSPP1 is about 10 times larger than that of FTSP, since DSPP1 has to be executed 10 times to compute the shortest paths for all time intervals. FD-Bellman is better than FTSP when the network size is small, since FD-Bellman does not have a distributed synchronizer in its execution. When the network size becomes large (i.e., $\geq$ 1K), FTSP outperforms FD-Bellman due to the exponential worst case message complexity of Bellman-Ford algorithms. We observed similar trends for time complexity for the three algorithms, those are omitted here.

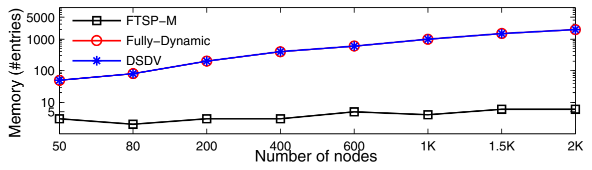
We also varied the time slots sets with a fixed network size of G_6, which represents medium-sized WSNs. The results are shown in Figure 5(b). We observe that FTSP and FD-Bellman do not change their message complexities since they only depend on the network size. The time cost for all the algorithms become worse when the average value of all elements in the selected time slots set becomes larger, since the average link delay is correspondingly increasing.

B. Route Maintenance

For evaluation of FTSP-M, we return to static networks by selecting the time slots set of C_1 in Table II. We do so for the purpose of fair comparisons with previous works in [12], referred to here as Full-Dynamic, and DSDV [24] with unicast support (multicast is not well-supported in duty-cycled WSNs).



(a) Different size of input change



(b) Memory required in each node

Fig. 6. Performance comparison for route maintenance

We first evaluate the effect of input changes on all algorithms for medium-sized networks by choosing G_6. Figure 6(a) shows the results. We observe that FTSP-M achieves the median message cost and time cost when input changes become large. The reason is that DSDV may suffer exponential message complexity, and FTSP-M uses the synchronizer which consumes additional time and needs more messages which is not as efficient as that in Full-Dynamic.

We also tested the average memory cost by varying network sizes. The results shown in Figure 6(b) indicate that FTSP-M achieves the best memory cost, which does not depend on the network size. The memory costs of DSDV and Full-Dynamic are increasing with network size, since each node stores an entry for all other nodes.

IX. Conclusions

In this paper, we addressed the distributed shortest path routing problem in duty-cycled WSNs. Our contributions are three-fold. First, we model duty-cycled WSNs as time-dependent networks, which satisfy the FIFO condition. Second, we present FTSP for finding shortest paths in such networks. FTSP have polynomial message complexity and are more time-efficient than previous solutions. Third, we present FTSP-M for distributed route maintenance with node insertion, updating, and deletion. FTSP-M are memory efficient and have polynomial message complexity. Directions for future work include investigating the time-dependent minimum spanning tree problem, which is NP-Hard in WSNs, and time-dependent multicast routing in WSNs, which may be the reverse direction of all-to-one routing.

References

[1] C. Schurgers and M. Srivastava, "Energy efficient routing in wireless sensor networks," in *Military Communications Conference (MILCOM 2001)*, vol. 1, 2001, pp. 357–361.

[2] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems (Sensys)*, 2003, pp. 14–27.

[3] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *International conference on Embedded networked sensor systems (Sensys'04)*, 2004, pp. 95–107.

[4] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *ACM Sensys*, New York, NY, USA, 2006, pp. 307–320.

[5] R. J., P. B., and C. V., "Adaptive low power listening for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 8, pp. 988–1004, 2007.

[6] Y. Gu and T. He, "Data forwarding in extremely low duty-cycle sensor networks with unreliable communication links," in *ACM SenSys 07*, 2007, pp. 32–38.

[7] K. L. Cooke and E. Halsey, "The shortest route through a network with time-dependent internodal transit times," *J. Math. Anal. Appl.*, vol. 14, pp. 493–498, 1966.

[8] A. Orda and R. Rom, "Distributed shortest-path protocols for time-dependent networks," *Distributed Computing*, vol. 10, no. 1, pp. 49–62, 1996.

[9] I. Chabini, "Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time," *Transportation Research Records*, vol. 1645, pp. 170–175, 1998.

[10] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *Proceedings of Extending database technology (EDBT'08)*, 2008, pp. 205–216.

[11] H. Chon, D. Agrawa, and A. Abbadi., "Fates: Finding a time dependent shortest path," *Mobile Data Management*, vol. 2574, pp. 165–180, 2003.

[12] S. C., G. D., D. F., and U. N., "A fully dynamic algorithm for distributed shortest paths," *Theoretical Computer Science*, vol. 297, no. 1-3, pp. 83–102, 2003.

[13] G. D'Angelo, S. Cicerone, G. Di Stefano, and D. Frigioni, "Partially dynamic concurrent update of distributed shortest paths," in *International Conference on Computing: Theory and Applications*, 2007, pp. 32–38.

[14] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[15] B. Awerbuch, "Complexity of network synchronization," *Journal of the ACM (JACM)*, vol. 32, no. 4, pp. 804–823, 1985.

[16] K. M. Chandy and J. Misra, "Distributed computation on graphs: shortest path algorithms," *Communications of the ACM*, vol. 25, no. 11, 1982.

[17] A. Segall, "Distributed network protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 1, pp. 23–35, Jan 1983.

[18] J. J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Trans. Netw.*, no. 1, pp. 130–141, 1993.

[19] R. Zheng, J. C. Hou, and L. Sha, "Asynchronous wakeup for ad hoc networks," in *ACM MobiHoc*, 2003, pp. 35–45.

[20] C. Vigorito, D. Ganesan, and A. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *IEEE SECON'07*, June 2007, pp. 21–30.

[21] K. V. Ramarao and S. Venkatesan, "On finding and updating shortest paths distributively," *J. Algorithms*, vol. 13, no. 2, pp. 235–257, 1992.

[22] OMNET++, http://www.omnetpp.org/.

[23] M. Zuniga and B. Krishnamachari, "Analyzing the transitional region in low power wireless links," in *IEEE SECON 2004*, 2004, pp. 517–526.

[24] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 234–244, 1994.