A distributed algorithm for finding maximum barrier coverage in wireless sensor networks

Jun He and Hongchi Shi
Department of Computer Science
Texas State University - San Marcos
San Marcos, Texas 78666
Email: {jhe216, hs15}@txstate.edu

Abstract—Constructing sensor barriers to detect intruders crossing restricted regions, such as country borders, is one of the major application categories for wireless sensor networks. In this paper, we present a distributed algorithm to find the maximum number of disjoint sensor barriers in wireless sensor networks. Our solution works for any sensor deployment, for any size and shape of a covered region, and even for heterogeneous sensor nodes. In particular, our algorithm is distributed and works perfectly in an asynchronous communication environment. It utilizes the property of wireless channel and has lower complexity compared with other algorithms. For a deployment of \( n \) sensors, our algorithm spends \( O(n^2) \) messages and \( O(n^2) \) time.

I. INTRODUCTION

Intrusion detection and border surveillance form a major application category for wireless sensor networks. The union of sensed areas of wireless sensors constitutes sensor barriers, which monitor the protected border. To detect the intrusion, the barrier needs to be strong or contains no gaps [1]. In this paper, we only consider the scenario that the barrier is strong.

If a region is covered by \( k \) disjoint sensor barriers, its coverage is referred to as \( k \)-barrier coverage. Compared with full coverage [2], which covers every point in the deployment region by sensors, barrier coverage requires much fewer sensors in wide deployment regions where it is not necessary to detect intruders at every point in their trajectory.

To know the quality of the deployment, the operator needs to find the maximum number of barriers after sensor deployment. If the number of barriers is less than the requirement, more sensors are needed. This problem is called maximum barrier coverage problem, defined as follows: A barrier is a set of sensor nodes that cover a connected strip of area between two sets of preassigned source and destination nodes at the edges of a region. Maximum barrier coverage is to find the maximum number of disjoint barriers in wireless sensor networks.

Recently, several papers have been published on the barrier coverage of wireless sensor networks [1], [3]–[6]. In [3], the notion of barrier coverage was first given by Kumar et al. in [1]. The authors proved that no algorithm can determine barrier coverage with \( O(1) \) computation time and also proposed a centralized algorithm to determine whether a region is \( k \)-barrier covered. However, since manual deployment usually is infeasible, the centralized algorithm is not good because it causes high communication overhead and computation cost that consume the precious energy quickly.

In [5], Chen et al. devised a localized algorithm to ensure that all crossing paths confined to a slice of the belt region of deployment would surely be detected. However, a crossing path that stretches to more than the length of the slice may not be detected by the localized barriers. Moreover, this localized algorithm cannot determine the maximum barrier coverage. In [6], the authors proposed a distributed algorithm that finds the barriers with minimum cost within a sensor network for a given number of barriers \( k \). It can be integrated to the algorithm presented in this paper to find the maximum number minimum cost of sensor barriers.

In [4], Liu et al. presented a distributed algorithm to construct sensor barriers in long strip areas. Their approach divides the original region into small segments interleaved by thin vertical strips. Locally found horizontal barriers are then connected by vertical barriers. However, their algorithm cannot find the maximum number of disjoint barriers in wireless sensor networks because each local segment only finds the local horizontal barriers. Moreover, their approach may turn on many unnecessary sensors to constitute vertical barriers.

In this paper, we propose an asynchronous distributed approach to the maximum barrier coverage problem. Our approach can quickly determine the quality of deployment in any shape or size of regions. The exact positions of sensor nodes are not necessary in our approach. To the best of our knowledge, our algorithm is the first to distributeally solve the maximum barrier coverage problem with \( O(n^2) \) time and message complexities.

The remainder of the paper is organized as follows. Section II gives a brief overview of network model and a brief summary of the Goldberg-Targan preflow-push algorithm. In Section III, our new PUSH-PULL algorithm is proposed. Its correctness and complexity are analyzed and discussed in Section IV. Section V concludes the paper.

II. OVERVIEW OF THE PROBLEM

A. Network model

Consider a wireless sensor network \( G \) with \( n \) sensor nodes. Let \( S = \{s_1, s_2, \ldots, s_{|S|}\} \) be the set of source nodes and \( T = \{t_1, t_2, \ldots, t_{|T|}\} \) be the set of destination nodes in \( G \). We assume a disc-based sensing and communication model where
an operating sensor node \( v \) has a sensing range of \( R_v^s \) and a communication range of \( R_v^c \). Node \( v \) can detect any moving target within the disc of radius \( R_v^s \) centered at \( v \). Any sensor node within the disc of radius \( R_v^c \) can receive the message from node \( v \). Note that in heterogeneous networks, sensor nodes may have different sensing and communication ranges.

Two sensor nodes \( u \) and \( v \) are neighbors to each other if the sensor disc of \( u \) intersects the sensor disc of \( v \). Let \( L_v \) be the set of neighbors of a sensor node \( v \). To guarantee neighboring communication, we assume \( R_v^c \geq R_u^s + R_v^c \), \( R_v^c \geq R_u^s + R_v^c \). In practice, the communication range is usually much larger than the sensing range, hence this condition is truly fulfilled. If node \( v \) has \( l \) neighbors \( L_v = \{ u_1, u_2, \ldots, u_l \} \), we can add 2\( l \) directed links (neighboring links) in network \( G \) to represent the communication links between \( v \) and \( u_1, u_2, \ldots, u_l \). We use \( G = (V, E) \) to represent the sensor network, where \( V \) is the set of sensor nodes and \( E \) is the set of neighboring links.

Each node \( v \) in \( G \) is divided into two sub-nodes \( v^a \), \( v^b \). A directed link with unit capacity from \( v^a \) to \( v^a \) is added, where unit node capacity enforces that any node allows at most one flow to go through it. The neighboring links are redefined by the links with unit capacity from \( u^a \) to \( v^b \) and from \( r^a \) to \( s^b \) if sensor nodes \( u \) and \( r \) are neighboring nodes in \( G \). Then we get a new network \( G' = (V', E') \), where \( V' \) is the set of nodes and \( E' \) is the set of links with unit capacity in \( G' \). Let \( n = |V| \) and \( m = |E| \), then \( N = |V'| = 2n \) and \( M = |E'| = 2m + n \). Consequently, maximum barrier coverage in \( G \) is converted to the maximum flow problem in \( G' \).

B. Maximum flow problem

In the 1990s, several algorithms for the maximum flow problem have been proposed for communication networks and other distributed systems [7], [8]. Our new algorithm is a significant improvement of the Goldberg-Tarjan preflow-push algorithm, a distributed algorithm with \( O(m n^2) \) message complexity and \( O(n^2) \) time complexity. Several basic concepts of maximum flow problem are given as follows.

**Definition II.1.** The preflow \( f \) between source node \( s \) and destination node \( t \) is a function \( f : V \times V \rightarrow \mathbb{R} \) such that \( f(v, w) \leq c(v, w) \forall v, w \in V, f(v, w) = -f(w, v) \forall v, w \in V, \) and \( \sum_{w \in V} f(v, w) \geq 0 \forall v \in V - \{ s, t \} \), where \( c(v, w) \) is the capacity of link \((v, w)\) and \( f(v, w) \) is the preflow passing through \((v, w)\). A valid flow is a preflow with \( \sum_{w \in V} f(v, w) = 0 \forall v \in V - \{ s, t \} \).

**Definition II.2.** The residual across two nodes \( v, w \in V \) is defined by function \( r : V \times V \rightarrow \mathbb{R} \) such that \( r(v, w) = c(v, w) - f(v, w) \), where the residual \( r(v, w) \) represents the amount of potential flow we can still push from \( v \) to \( w \).

**Definition II.3.** A valid labeling is a function \( D : V \rightarrow \mathbb{Z} \) with properties: \( D(s) = n, D(t) = 0, \) and \( D(v) \leq D(w) + 1 \forall (v, w) \in E_R, \) where \( E_R = \{(v, w) \in V \times V | r(v, w) > 0\} \). \( D(v), v \in V - \{ s \}, \) represents the lower bound distance from node \( v \) to destination \( t \) in residual network \( G = (V, E_R) \).

**Definition II.4.** The excess of \( v \in V \) is a function \( e : V \rightarrow \mathbb{R} \) and \( e(v) = \sum_{w \in V} f(w, v) \). For any valid flow, \( e(v) = 0 \forall v \in V - \{ s, t \} \). For any valid preflow, \( e(v) \geq 0 \forall v \in V - \{ s, t \} \). A node \( v \in V - \{ s, t \} \) is active if and only if \( e(v) > 0 \).

For the sake of completeness, we give the complexity of pre-flow algorithm from [7], [9] without proof. To bound the running time of the algorithm in asynchronous distributed systems, we introduce the concept of time unit [9].

**Definition II.5.** A time unit is the longest time from a time point when a message is originated by a sender to the time point when this message is processed by its receiver.

**Lemma II.1.** The asynchronous distributed implementation of preflow-push algorithm runs in \( O(|V|^2) \) time using \( O(|V|^2|E|) \) messages in network with \( |V| \) nodes and \( |E| \) links.

**Corollary II.2.** The asynchronous distributed implementation of preflow-push algorithm runs in \( O(|V|^2) \) time using \( O(|V||E|) \) messages in network with unit node capacity.

III. PROPOSED PUSH-PULL ALGORITHM

In this section, we present our fast distributed algorithm. We assume that the underlying communication protocol in wireless sensor networks provides a communication channel that guarantees every message reaches its destination with arbitrary but finite delays. Any message from sensor node \( v \) has a large chance to be received by its neighbors. Acknowledgment messages are ignored in the description of the algorithm.

A. Local variables and message structures

Node \( v \) locally stores \( D_v^b \) and \( D_v^a \), the distance labels for \( v^b \) and \( v^a \), and a neighboring list \( L_v = \{ u | \forall (u, v) \in E \} \), where the neighbors are confined by the sensor range instead of communication range. Let \( D_v^{out}(u), \forall u \in L_v \), store neighbors’ distance labels \( D_v^b \), \( \forall u \in L_v \). The usage of input and output of \( v \) is marked by two indicators, \( C_v^{in} \) and \( C_v^{out} \), stores the neighboring node that occupies \( v^a \)’s input/output, or 0 to indicate that input/output is available. \( D_v^{in} \) represents the distance label of node \( C_v^{in} \). An indicator \( A_v \) indicates whether node \( v \) is active \((A_v = 1)\) or inactive \((A_v = 0)\). A variable node_type distinguishes sensor nodes, with 1 for source node, 2 for destination node, and 3 for other nodes. The source and destination nodes can be designated before or after deployment. If these nodes are designated after deployment, the base station needs to inform nodes residing near the edges and configure them as source or destination nodes.

There are two types of messages (PUSH and PULL), whose structures are both shown in Fig. 1. PUSH message is sent from node \( u \)’s sub-node \( u^a \) to another node \( v \)’s sub-node \( v^b \), \( \exists (u, v) \in E_R \) and PULL message is sent from node \( v \)’s sub-node \( v^b \) to another node \( u \)’s sub-node \( u^a \), \( \exists (v, u) \in E_R \). Due to the property of the wireless channel, every neighbor may hear a message even though their IDs are not equal to dest_ID. If BRST field is 1, neighbors will locally update local variables based on the distance label values in the broadcast message from node src_ID.
Fig. 1. Structures of PUSH and PULL messages. The first bit of messages is used to indicate the message type, with 0 for PUSH and 1 for PULL. The second bit indicates if it is a broadcasting request.

B. The new distributed PUSH-PULL algorithm

The details of the algorithm including subroutines are shown in Algorithm 1, Procedures 1 & 2, and Process MSGs 1 & 2.

Algorithm 1 PUSH-PULL approach executed at node \( v \)

1: Initialization();
2: while 1 do
3:   3: Wait for an incoming message (MSG) and process it
4:   end while

After deployment, every sensor node boots up and initializes based on Procedure 1. When a sensor node \( v \) receives a message from node \( u \), \( v \) determines if \( u \) is one of its neighboring nodes. There are many techniques available for this in literature, such as measuring the received signal strength [10]. We assume that every node \( v \) will finally find all of its neighbors and then save the IDs in \( L_v \). In heterogenous sensor networks, \( v \) may also broadcast the value of \( v \)'s sensing range and the transmitting signal power.

Procedure 1 Initialization() at node \( v \)

1: Initialize node_type, \( A_v = 0 \), \( v^a = 1 \), and \( v^b = 2 \)
2: Broadcast its identity
3: Discover all neighboring nodes
4: Assign initial values (0) to other local variables
5: if node_type == 1 then
6:   Add \( s' \) to \( L_v \) and assume itself receives a message, MSG(PUSH, 0, \( v \), \( s' \), \( 2n + 3 \), \( 2n + 4 \))
7: else if node_type == 2 then
8:   Add \( t' \) to \( L_v \), \( D_v^{out}(t') \) = 0
9: end if

After initialization, source nodes will send PUSH message to every possible neighboring node. When node \( v \) receives a PUSH message from node \( u \), it follows the steps in Procedure 1. If \( v \) is an inactive intermediate node, it will push this preflow to one of its neighboring nodes with \( D_v^{in} = D_v^{out}(w) + 1 \), \( w \in L_v \). If no such node exists, \( v \) will lift its distance label to a certain level to either push forward or pull backward based on Procedure 2.

In LIFT_PUSH_PULL procedure, node \( v \) first finds the neighbor node \( w \) with \( \min_{w \in L_v} D_v^{out}(w) \). If \( w^b \)'s distance label is less than \( w^a \)'s distance label minus 1, \( v \) will update its distance labels and push the preflow forward because \( w \) has a smaller distance label to destination and it is more possible that \( v \) reaches destination by a shorter path through \( w \). If not, this preflow will be pulled back to \( u \) because \( v \) may not be the correct next stop for preflow in the shortest path to \( T \).

If an active node \( v \) receives a PUSH message, \( v \) will pull back either previous flow or current preflow. If \( \nu^b \)'s distance label is larger than \( \nu^a \)'s (saved in \( D_v^{in} \) at node \( v \)) distance label, the previous flow will be pulled back. Otherwise, the current preflow will be pulled back to its sender \( u \).

Because of delays, \( u \) sometimes stores an outdated distance label of \( v \) and sends a PUSH message to \( v \). \( v \) will discover this error by checking \( u \)'s distance label in PUSH message. The error occurs if \( \nu^a < D_v^{in} + 1 \) and then \( v \) will send a PULL message with its current distance label to \( u \).

Process MSG 1 On receipt of MSG(PUSH, BRST, dest_ID, \( v \), \( D_v^a \), \( D_v^b \))

1: if \( v \) in \( L_u \) & & \( u \) \( \neq \) dest_ID & & BRST == 1 then
2:   \( D_v^{out}(v \) \) = \( D_v^b \) if \( v \) \( \neq \) \( D_v^b \)
3: else if \( v \) = = dest_ID then
4:   \( D_v^{out}(v \) \) = \( D_v^b \) if \( v \) \( \neq \) \( D_v^b \)
5: if \( v \) \( \neq \) \( D_v^a \) \&\& \( v \) \( \neq \) \( D_v^b \) \&\& \( v \) \( \neq \) \( D_v^a \) then
6:   \( A_v = 0 \)
7: if \( w \) \( \neq \) \( \nu \) \&\& \( \nu^a = D_v^{in} \) then
8:   \( D_v^{in} = D_v^{in} + 1 \)
9: if \( \exists w \) \( \in L_v \) & & \( \nu^a = D_v^{in} + 1 \) then
10:   \( D_v^b = D_v^b + 1 \)
11: MSG(PULL, 0, \( w \), \( v \), \( D_v^b \))
12: end if
13: else
14: MSG(PULL, 0, \( v \), \( D_v^b \))
15: end if

Process MSG 2 On receipt of MSG(PULL, BRST, dest_ID, \( v \), \( D_v^a \), \( D_v^b \))

1: if \( v \) in \( L_u \) & & \( v \) \( \neq \) dest_ID & & BRST == 1 then
2: \( D_v^{out}(v \) \) = \( D_v^b \) if \( v \) \( \neq \) \( D_v^b \)
3: else if \( v \) = = dest_ID then
4: \( D_v^{out}(v \) \) = \( D_v^b \) if \( v \) \( \neq \) \( D_v^b \)
5: if \( \exists w \) \( \in L_v \) - \( C_v^a \) & & \( \nu^a = D_v^{in} + 1 \) then
6: MSG(PULL, 0, \( w \), \( v \), \( D_v^a \))
7: \( C_v^{out} = w \)
8: else
9: LIFT_PUSH_PULL();
10: end if
11: end if

When a PULL message from \( w \) arrives at node \( v \), \( v \) checks if \( w \) is in its neighboring list and determines if it should receive the message. If no, \( v \) will check the BRST bit to see whether it should update \( w \)'s distance labels or not. If yes, \( v \) will update
the local distance value of \( w \). Let \( u \) be the previous node of \( v \) in this preflow. Node \( v \) will try to find a different sensor node \( w' \) with \( D^u_v = D^\text{out}_v(w)+1 \) and then push the preflow to \( w' \) again. If \( v \) cannot find any \( w' \), \( v \) will run \text{LIFT}_{\text{PUSH}_{\text{PULL}}} \) procedure to decide whether it sends a PUSH message to one of its neighboring nodes or sends a PULL message to \( u \).

**Procedure 2 LIFT\_PUSH\_PULL()**

1. choose \( w \) with \( \min_{u \in L_v} D^\text{out}_u(w) \);
2. if \( D^\text{out}_v(w) < D^v_v - 1 \) then
3. \( D^a_v = D^\text{out}_v(w) + 1 \);
4. if \( D^a_v > D^b_v \) then
5. \( D^b_v = D^a_v + 1 \);
6. MSG(PUSH, 1, w, v, \( D^a_v \), \( D^b_v \));
7. else
8. MSG(PUSH, 0, w, v, \( D^a_v \), \( D^b_v \));
9. end if
10. if \( A_v = 0 \) then
11. \( A_v = 1 \);
12. end if
13. \( C^\text{out}_v = w \);
14. else
15. \( D^a_v = D^\text{in}_v \);
16. if \( D^a_v > D^b_v \) then
17. \( D^b_v = D^a_v + 1 \);
18. MSG(PULL, 1, \( C^\text{in}_v \), v, \( D^b_v \));
19. else
20. MSG(PULL, 0, \( C^\text{in}_v \), v, 0);
21. end if
22. if \( A_v = 1 \) then
23. \( A_v = 0 \); \( C^\text{out}_v = 0 \);
24. end if
25. \( C^\text{in}_v = 0 \); \( D^\text{in}_v = 0 \);
26. end if

The PUSH-PULL algorithm will terminate if \( \sum_{s \in S} A_s = \sum_{t \in T} A_t \). Subsequently, all active sensor nodes (\( A_v = 1 \)) will form the maximum barrier coverage and all inactive sensor nodes (\( A_v = 0 \)) will go into sleeping mode.

IV. CORRECTNESS AND COMPLEXITY ANALYSIS

A. Correctness and termination

**Lemma IV.1.** The algorithm maintains the invariant that \( D \) is a valid labeling and all distance labels never decrease.

**Proof** We use induction to prove it. Initially, \( D^a_v = 1, D^b_v = 2, \forall v \in V - \{s', t'\}, D^\text{out}_v = 2n + 3, D^\text{in}_v = 0 \), thus \( D \) is a valid labeling. Considering that \( v \in V - \{s', t'\} \) sends a PUSH or PULL message to \( u \), this may add (\( u, v \)) link to \( E_R \) and delete (\( u, v \)) from \( E_R \). Because \( D^a_v = D^b_v + 1 \) and \( D^b_v = D^a_v - 1 < D^a_v + 1 \), the addition of (\( u, v \)) (from \( u \) to \( v \)) to \( E_R \) does not affect the validity of \( D \). In addition, \( D^b_v = D^a_v + 1 \) if \( v \) is inactive and \( D^b_v < D^a_v + 1 \) if \( v \) is active. Thus, the execution maintains the invariant that \( D \) is a valid labeling. The receipt of PUSH/PULL message never decreases the label values. Thus distance labels never decrease. 

**Lemma IV.2.** For any valid labeling \( D \), the source node \( s' \) and the destination nodes \( t' \) must be disconnected in residual network \( G_R = (V, E_R) \).

**Proof** For the sake of contradiction, assume there exists a path \( P = s''_1, v^1, v^2, \ldots, v^n, t' \). Each node is visited once in the path, so that \( p \leq n \) and \( |P| = 2p + 2 \leq 2n + 2 \). For any link (\( u, w \)) in \( P \), the distance label of \( u \) must be no greater than the distance label of \( w \) plus 1. So \( D(s'') \leq D(t') + |P| - 1 \leq 2n + 1 \), which contradicts \( D(s'') = 2n + 3 \).

**Lemma IV.3.** For any node \( v \in V - \{s', t'\}, D^a_v = 2n + 3 \) and \( D^b_v = 2n + 4 \).

**Proof** Initially, \( D^a_v = 2n + 3 \) and \( D^b_v = 2n + 4 \). At some point, assume that \( D^a_v \) at node \( v \in V \) is the first one larger than \( 2n + 3 \) among all distance labels. According to the PUSH-PULL algorithm, \( D^a_v \) can only be increased by Lines 3 and 15 of Procedure 2. If by Line 3, \( D^\text{out}_v(w) = D^a_v - 1 > 2n + 2 \), then \( D^\text{in}_v > D^\text{out}_v(w) + 1 > 2n + 3 \) based on Line 2. If by Line 15, \( D^b_v = D^\text{in}_v + 2n + 3 \). Both cases infer that \( D^a_v = D^\text{in}_v + 2n + 3 \), \( \exists u \in L_v \). It contradicts our assumption that \( D^a_v \) is the first distance label larger than \( 2n + 3 \). So \( D^a_v \leq 2n + 3 \). \( D^b_v \) can only be modified by Lines 12 and 18 of Process MSG 1. Line 12 calls Procedure 2, where \( D^b_v \) can be increased to \( D^a_v + 1 \) by its Lines 5 and 17. Line 18 makes \( D^b_v = \text{var}_{D^a_v} + 1 \). Since \( D^a_v \leq 2n + 3 \), Lines 12 and 18 of Process MSG 1 cannot make \( D^b_v > 2n + 4 \). Thus \( D^b_v \leq 2n + 4 \).

**Lemma IV.4.** If there are no more PUSH-PULL messages, the PUSH-PULL algorithm terminates and the maximum barrier coverage is formed.

**Proof** If there are no more PUSH-PULL messages, the PUSH-PULL algorithm terminates because it is driven by the event of the receipt of messages. Moreover, it infers that each node \( v \in V - \{s', t'\} \) must have zero excess because \( v \) has unit excess temporarily when it receives a PUSH or PULL message. A flow \( f \) (node-disjoint paths) has been formed by all of active nodes (having flow passing through). Based on Lemma IV.2, \( s' \) and \( t' \) are disconnected in residual network \( G_R = (V, E_R) \). It is the classic condition that the total flow \( \| f \| \) is maximum. Thus the maximum barrier coverage is formed by the active sensor nodes which form the flow paths.

**Corollary IV.5.** PUSH-PULL algorithm terminates and the maximum barrier coverage is formed if \( \sum_{s \in S} A_s = \sum_{t \in T} A_t \) and the maximum number of barriers is equal to \( \sum_{s \in S} A_s \).

**Proof** When any node \( v \in V - \{s', t'\} \) receives and processes a message, it must also originate a new message and send it to one of its neighbors. The number of PUSH and PULL messages in the network at any time is no greater than the number of initial PUSH messages \( (F_m = |S|) \) from source node \( s' \). If any message arrives at \( t' \) through any destination node \( t \) in \( T \), it must be a PUSH message; the number of messages in the network is decreased by 1, and \( \sum_{t \in T} A_t \) is increased by 1. If any message is sends to \( s' \) through any source node, it must be a PULL message; the number of messages in the network...
Thus, there are at most \( n \) received by nodes is \( n \). The total number of PUSH messages is at most \( n \) because it is the maximum number of flow \( |f| \) in the network.

### B. Complexity analysis

**Lemma IV.6.** The total number of PUSH messages sent is no less than the total number of PULL messages sent.

**Proof** Based on the definition of the PULL operation, any PULL message is generated to pull a preflow back to its previous node along the trace of preflow. It infers that we can find a previous PUSH message from \( w \) to \( v \) for any PULL message from \( v \) to \( w \). Thus, the total number of PUSH messages sent is no less than that of PULL messages sent.

**Lemma IV.7.** The algorithm sends \( O(n^2) \) push messages and \( O(n^2) \) pull messages.

**Proof** Let’s first consider that all PUSH messages are sent by correctly knowing the distance labels of the neighboring nodes. Any push message arrives at a node \( v \in V - \{s'\} \) when \( v \) is either inactive or active. If \( v \) is inactive and the received PUSH message is directly pulled back and \( v \) still stays inactive, the label \( D^o_v \) during this process will be increased to \( D^o_v \), which infers \( D^o_v \) is increased by 2. Since \( 1 \leq D^o_v \leq 2n + 3 \), \( v \) can receive at most \( n + 1 \) PUSH messages before termination if PUSH messages do not make \( v \) active. If \( v \) is inactive and the received PUSH message changes \( v \)’s flag into active, the labels at \( v \) during the change of its flag from inactive to active may not change. But the labels at \( v \) during the change of its flag from active to inactive must be changed, and \( D^o_v \) will be increased by at least 2 because \( D^o_v = D^o_v \) now. Hence, there are at most \( n + 1 \) times that \( v \)’s status from inactive to active before termination because of \( 1 \leq D^o_v \leq 2n + 3 \) from Lemma IV.3. From the above analysis, we can see that at most \( (n + 1) \) PUSH messages are received at node \( v \) when \( v \) is inactive. Thus, there are at most \( n(n + 1) \) PUSH messages arriving when nodes are inactive.

If \( v \) is active, the received PUSH message will change the distance label \( D^b_v \) by at least 2 because either \( D^b_v = D^o_v + 1 \) or \( D^b_v = var(D^o_v) + 1 \). Thus, before termination, the number of received PUSH messages is at most \( n + 1 \) when \( v \) is active. Thus, there are at most \( n(n + 1) \) PUSH messages arriving when nodes are active. The destination nodes receive at most \( n \) PUSH messages. The total number of PUSH messages received by nodes is \( 2n^2 + 3n \) or \( O(n^2) \).

If PUSH messages are sent by using outdated information, the number of erroneous PUSH and PULL messages because of asynchonous transmission is bounded by the number of labeling changes in the network, which is \( O(n^2) \). Thus, the algorithm sends \( O(n^2) \) push messages. From Lemma IV.6, we have the fact that the algorithm sends \( O(n^2) \) pull messages.

**Lemma IV.8.** The PUSH-PULL algorithm uses \( O(n^2) \) messages in the sensor network with \( n \) sensor nodes.

**Proof** There are \( O(n^2) \) PUSH, \( O(n^2) \) PULL, and \( O(n^2) \) acknowledgment messages received by \( n \) sensor nodes. Non-destination nodes do not need to send acknowledgment to the source node when \( BRST = 1 \). The failure of receiving broadcasting messages may cause wrong PUSH or PULL action. Then the sender will receive an update PULL or PUSH message including new distance labels. These update messages are bounded by the number of price changes \( O(n^2) \). Thus, PUSH-PULL algorithm uses \( O(n^2) \) messages.

**Lemma IV.9.** The PUSH-PULL algorithm runs in \( O(n^2) \) time in the sensor network with \( n \) sensor nodes.

**Proof** It is obvious that in the worst case, the distributed algorithm runs sequentially and each message takes 1 unit of time. Thus, the PUSH-PULL algorithm runs in \( O(n^2) \) time.

**V. CONCLUSION**

We present a distributed PUSH-PULL algorithm that finds maximum barrier coverage in any wireless sensor network. The PUSH-PULL algorithm utilizes the broadcast properties of wireless communication channels and node disjointness, reducing the message complexity to \( O(n^2) \). Our method significantly reduces the message complexity compared with the preflow-push algorithm. Thus, the energy consumption for building up maximum barrier coverage is greatly decreased.

**REFERENCES**


