An Efficient Graph Search Algorithm for Backbone Discovery In Wireless Linear Sensor Networks

Imad Jawhar[1], Jie Wu[2], Nader Mohamed[1] and Sheng Zhang[3]

[1]College of Information Technology, UAE University, Alain, UAE
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, Pennsylvania, USA
[3]State Key Laboratory for Novel Software Technology, Nanjing University, P. R. China

***Abstract*— Wireless sensor networks (WSNs) is an area of research that has been getting a lot of attention lately. This is due to the rapid advancements in the design of wireless devices which have increasingly more processing, storage, memory, and networking capabilities. In addition, the cost of sensors is constantly decreasing making it possible to use large quantities of these sensors in a wide variety of important applications in environmental, military, commercial, health care, and other fields. In order to monitor certain types of infrastructures, many of these applications involve lining up the sensors in a linear form, making a special class of these networks which are defined in this work as Linear Sensor Networks (LSNs). In a previous paper, we introduced the concept of LSNs along with a classification of the different types of LSNs, a sample of their applications and the motivation for designing specialized protocols that take advantage of the linearity of the network to enhance their communication efficiency, reliability, fault tolerance, energy savings, and network lifetime. This paper presents a graph-search-based topology discovery algorithm for LSNs. New definitions for important structure and design parameters are introduced. The proposed protocol allows the nodes to identify some nodes to be included in a backbone, which can be used by the other nodes to send data to the sink at the end of the LSN or LSN segment. This backbone discovery increases the efficiency, and robustness of the network. It also allows for significant improvement in the scalability of the communication process in the LSN which can contain a very large number of nodes (e.g. hundreds or thousands). In addition, linearity of the structure and discovered backbone can enhance the routing reliability by "jumping" over failed nodes by increasing the range. Furthermore, the protocol does not require the nodes to have location detection capabilities such as GPS, which would lead to a more complex design and higher cost of the sensor nodes.**

Keywords: Ad hoc and sensor networks, routing, backbone discovery, wireless networks.

I. Introduction

Wireless Sensor Networks (WSNs) have received a lot of attention due to constant advancements in the field of electronics and wireless communication which have led to the design of low cost, small, and capable sensing devices with increasingly higher processing, storage, sensing and communication capabilities. In addition, WSNs have a great potential for use in a large amount of existing and future applications in numerous areas such as environmental, civil, health care, military, monitoring, and infrastructure surveillance. In the latter category, a considerable number of the infrastructures that are monitored have a linear structure which extends over relatively long distances. This causes the wireless sensors to be aligned in a linear topology. New frameworks and protocols are needed to take better advantage of the linearity of the network structure in order to increase routing efficiency, enhance reliability and security, and improve location management. In a previous paper [1], we introduced a classification of LSNs from a hierarchical and topological points of views.

In this paper, we introduce a topology discovery algorithm for thick LSNs, where the sensor nodes are deployed between two parallel lines that can stretch for a long distance (e.g. tens or hundreds of kilometers). As a result of the proposed topology discovery algorithms, a small percentage of the deployed sensor nodes are selected to form a backbone network along the linear topology, which can be used to efficiently route sensing data (collected from the surrounding nodes and transmitted to the nearest backbone node) along the linear network to the sink or sinks located at the end of the network or network segment.

The characteristics of one-dimensional ad hoc networks have been studied by various researchers. Diggavi et. al. studied the characteristics of wireless capacity with the existence of mobility in one-dimension [2]. Ghasemi et al. provided an approximation formula for the connectivity probability of one-dimensional ad hoc wireless networks [3]. Miorandi et al. analyzed the connectivity issue in one-dimensional ad hoc networks using a queuing theory approach [4]. On the other hand, many researchers have investigated topology control (TC) techniques in wireless ad hoc networks. In [5], Santi et al. present a survey of these algorithms, which have the primary goal of reducing energy consumption, and radio interference. In [6], Ramanathan et al. study the optimization problem of creating a desired topology by adjusting the transmit power of the nodes. In another paper [7], the authors study power assignments to maintain fault tolerance in wireless devices and present algorithms which can be used to minimize power while maintaining $k - edge$ connectivity with guaranteed approximation factors. In [8], a topology discovery algorithm for WSNs is presented. The algorithm determines a set of nodes which can act as cluster heads in the network. In [9], Wang presents an overview of the different types of topology algorithms for multidimensional WSNs that have

This work was supported in part by the UAEU Program for Advanced Research (UPAR), grant No. 31T045.

been proposed in research.

The algorithms that are mentioned above are primarily designed for multi-dimensional WSNs. They do not take advantage of the predictable topology of a thick LSN in order to optimize their performance. On the other hand, the algorithms presented in this paper are designed to take advantage of the linearity of the network in order to reduce topology discovery control overhead, and increase operation efficiency, and scalability.

In a thick LSN, the sensor nodes have the responsibility of both sensing the information as well as routing it through their neighbor nodes along the "thick line" of sensor nodes to finally reach the sink node at the end of the network. In this case, the sensor nodes have sensing, aggregation, compression, as well as routing responsibilities. The thick LSN topology can be present in many applications such as the case when the WSN is responsible for monitoring a geographic area. For example, the network can have the responsibility of monitoring international borders between countries [10] and detect illicit activities. Such activities can involve border crossings by smugglers of different illegal goods or substances, military crossings by individuals, or vehicles, etc. The inexpensive sensors can be deployed by throwing them from an airplane moving at a constant but low speed or an unmanned aerial vehicle (UAV). The dropped sensors end up in a semi-random geographic form and could follow a linear structure. The sink nodes can also be deployed at various locations and are separated by some specified average distance. This deployment of the sink nodes can be done in many different ways. They could also be thrown from a low-flying airplane, placing them at locations which are separated by approximately the same average distance, or they can be installed [9] in a precise fashion by the network personnel if the terrain is easily accessible. Applications for linear sensor networks include but are not limited to the following: (1) Above-ground oil, gas, and water pipeline monitoring. (2) Underwater oil, gas, and water pipeline monitoring. (3) Railroad/subway monitoring. (4) Terrestrial border monitoring. (5) Sea-coast monitoring. (6) River monitoring.

We can identify various reasons why a new framework and architecture are needed for different categories of thick LSNs. Such reasons include: (1) Speed-up route the route discovery and maintenance. (2) Reduce control overhead and bandwidth utilization for route discovery. (3) Increased routing fault tolerance and reliability. (4) Reduce control overhead for route maintenance. (5) Increased efficiency of location management.

The rest of the paper is organized as follows. Section II presents the topology discovery algorithm for thick LSNs. Section III provides simulation results and analysis of some aspects of the discovery process, and section IV concludes the paper.

II. Topology Discovery Algorithm

A. Linear Backbone Discovery (LBD) Algorithm

The LBD algorithms is shown in Algorithms 1, 2, 3, and 4. It is also illustrated in Figure 1. It works in the following

Algorithm 1 Backbone Discovery - Initialization of Node Discovery Variables and Broadcasting of the LD message From the First Node at the Primary Edge

$myColor = WHITE$
/* set my temporary parent and my confirmed parent equal to ϕ. */
$myTempParent = myConfParent = \phi$
if (this is the first node at the primary edge) **then**
  /* First node in the list. So, set myLC to 0. */
  $myLc = 0$
  /* Initiate the discovery process by sending the first LD message. */
  $messageLc = 1$
  /* Initialize the discovered $PATH$ list to $myID$. */
  $PATH = myID$
  /* Broadcast LD message to all neighbors. Only the first node starts by sending an LD message */
  $SendLD(messageID, myID, messageLc, PATH)$ to all neighbors
else
  /* Ordinary node. So, initialize myLC to ∞ and wait for an LD message to update myLC. */
  $myLc = \infty$
end if

Algorithm 2 Backbone Discovery - Algorithm at an Intermediate Node y When Receiving an LD Message From a Node x

/* Note: the distance (in number of hops) while the LD message is propagating is the distance from the node that initiated the discovery process at the primary edge of the LSN. */
When node y receives the $LD(messageID, x, messageLc)$ from node x it does the following:
if $(messageLc < myLc)$ **then**
  /* Distance in message is better. So, the distance can be relaxed further. Note that if y is $WHITE$ then $myLc = \infty \Rightarrow messageLc < myLc$ */
  $myTempPrarent = x$
  /* Set $myLc$ counter (i.e. distance of y) to lc. */
  $myLc = messageLc$
  $messageLc = messsageLc+1$
  /* Add $myID$ to the discovered $PATH$ list. */
  $PATH = PATH \mid myID$
  /* Broadcast the LD message to all neighbors */
  Broadcast $LD(messageID, myID, messageLc\ , PATH)$
else
  /* Distance of received message is not better than current distance set by a previous message. */
  Drop LD message
end if

Algorithm 3 Backbone Discovery - Algorithm at the Sink when Receiving a Linear Discovery LD Message From a Node x

When the sink receives the $LD(messageID, x, messageLc, PATH)$ message from node x it does the following:
$myBacwadNeigh=x$
/* Save the length of the backbone in number of hops and send it in the SF message. */
$BBLc = messageLc$
/* Send a sink found SF message to the backward direction neighbor x. Note the SF message is unicast back to the backward direction neighbor. */
Send $SF(messageID, source = myID, destination = x, BBlc, PATH)$

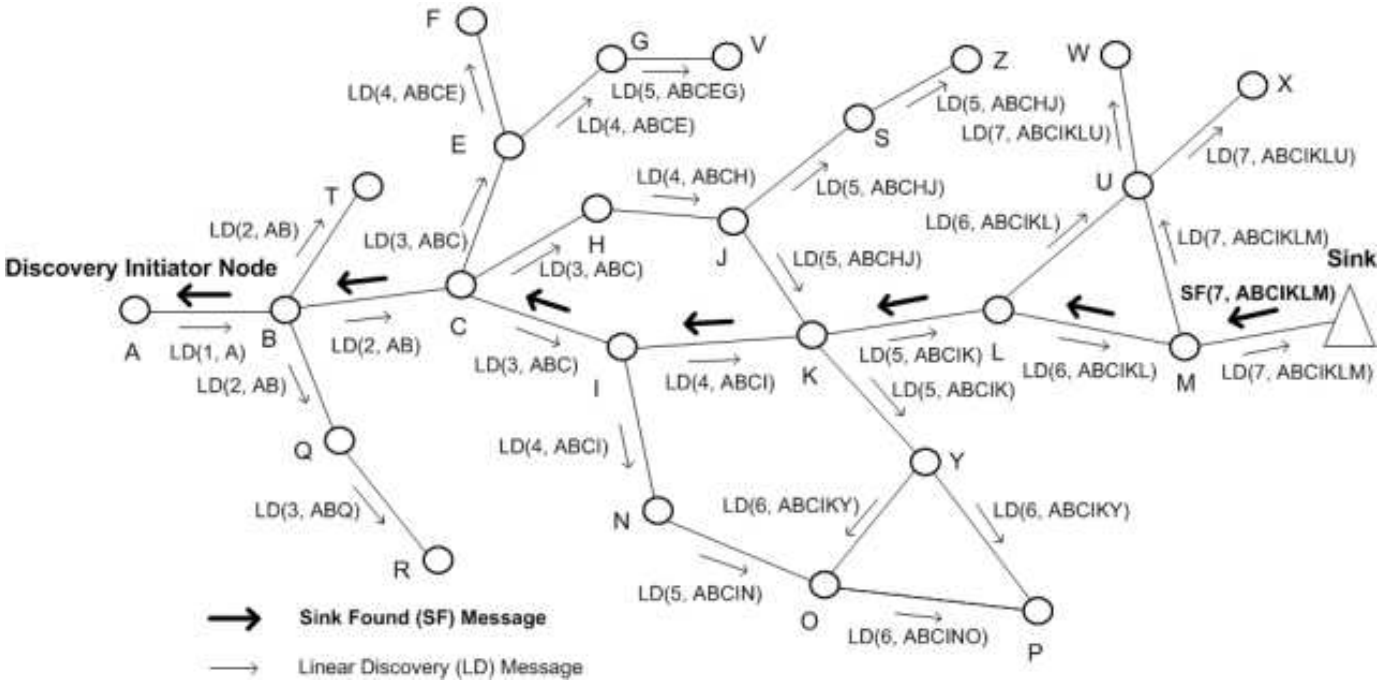


Fig. 1: Illustration of the LD message propagation from the initiator node to the sink in the Linear Backbone Discovery (LBD) algorithm.

manner. As indicated in Algorithm 1, the designated first node on the primary edge of the LSN starts the discovery process by initializing its discovery variables and broadcasting the Linear Discovery (LD) message $LD(messageID, myID, messageLc, PATH)$ to all of its neighbors. The message contains the following parameters:

1) $messageID$: This is the ID or the discovery message to prevent looping.
2) $myID$: This is the ID of the sending node.
3) $messageLc$: This is the linear discovery counter. It holds the count of nodes in the discovered path from the initial primary edge node that initiated the discovery process.
4) $PATH$: This is an ordered list of nodes that are contained in the discovered path.

Algorithm 2 describes the actions executed by an intermediate node y when it receives an $LD(messageID, myID, messageLc, PATH)$ message from node x. First node y checks to see if the linear counter in the message $messageLc$ is better (i.e. smaller) than its own linear counter $myLc$. If that is the case, then it changes its temporary parent to x, and updates its $myLc$ counter with that, which is in the message. It then increments the linear counter in the message by one, adds its own ID to the $PATH$ and broadcasts the updated $LD(messageID, myID, messageLc, PATH)$ message to all of its neighbors. However, if the linear counter in the message, $messageLc$ is not smaller than it own linear counter, $myLc$, then it drops the message since this implies that it already has a parent node with a better linear count with a smaller number of hops from the source, which contributes with a lower number of hops in the backbone.

Algorithm 3 describes the actions taken by the sink when it receives the $LD(messageID, myID, messageLc, PATH)$ from a node x. Namely, it saves the ID of x as its backward neighbor as well as the length of the discovered backbone in number of hops that is contained in the $messageLc$. It then unicasts a sink found message, $SF(messageID, source = myID, destination = x, BBlc, PATH)$ back to the discovery initiating node through the nodes in the discovered backbone.

Algorithm 4 describes the steps taken by an intermediate node y when it receives an $SF(messageID, source = myID, destination = x, BBlc, PATH)$ message from a node x. First, y sets its $iAmPartOfBackbone$ variable to $TRUE$. Then node y fully or partially caches the discovered backbone depending on the strategy that is used. A full caching of the backbone allows the node to have the full list of the nodes in the backbone and consequently node y has more flexibility in routing packets. However, this comes at the cost of increased memory usage. On the other hand, node y can partially cache the local part of the backbone such as k nodes in each direction, which allows it some flexibility in routing packets and reaction to neighboring node failures while reducing its memory usage. Node y then sets its forward and backward direction neighbors, as well as the distance from the source, and distance from the sink in number of hops. Afterwards, node y forwards the SF message to its backward neighbor. This propagation of the SF message continues along the nodes in the discovered backbone till it reaches the source node, thereby completing the backbone discovery process.

At the end of the backbone discovery process, we will have two types of nodes:

- *Backbone Nodes (BNs)*: These nodes are a part of the

backbone.

- *Non-backbone Nodes (NBs)*: These nodes did not end up being a part of the backbone. They are used to perform normal sensing operations.

Algorithm 4 Backbone Discovery - Algorithm at an Intermediate Node y When Receiving a Sink Found SF Message From a Node x

When node y receives the $SF(messageID,\ x,\ messageLc,\ PATH)$ message from node x it does the following:
/* Confirm being a part of the discovered backbone, and cache the discovered backbone in $PATH$ in the routing table. */
$iAmPartOfBackbone = TRUE$
Save the full or local part of the discovered backbone in $PATH$ in the routing table according to the adopted backbone caching strategy.
/* In this paper's current strategy, we save full $PATH$ */
$myBackwardDirNeigh = myTempParent$
$myForwardDirNeigh = x$
$myDistFromSource = messageLc$
$myDistFromSink = messageLc - myLC$
/* Forward message to backward direction neighbor. Note the SF message is unicast back to the backward direction neighbor. */
Send $SF(messageID,\ source = myID,\ destination = x,\ messageLc,\ PATH)$

Algorithm 5 NBD Initiation - Algorithm initiated by a newly declared BN node

/* Set the $sourceBNID$ to the ID node that is initiating the broadcast of the NBD message. */
$sourceBNID = myID$
/* Set the ring size of the NBD message propagation. */
$NBDringSize = \rho$
/* Initialize the number of hops from BN to 0 */
$numOfHops = 0$
/* Initialize $PATH\_TO\_BN$ list to only contain the ID of the current node. */
$PATH\_TO\_BN = myID$
/* Broadcast NBD message to all neighbors. */
Broadcast NBD $(messageID,\ sourceBNID,\ myID,\ NBDringSize,\ numOfHops,\ PATH\_TO\_BN)$

B. The New BN Declaration (NBD) Broadcast algorithm

At the end of the backbone discovery process, the newly discovered BN nodes will broadcast a a New BN Declaration (NBD) message to inform all of the nodes within ρ hops from itself that it is a part of the backbone. Algorithm 5 is used by the BN node to initiate the broadcast process of the NBD message. In the NBD message, the BN node includes the following parameters:

- $messageID$: This variable contains the message ID to prevent looping.
- $sourceBNID$: This is the ID of the sending BN node.
- $myID$: This is the ID of the node forwarding the BND message. Initially, it is equal to the $sourceBNID$.
- $BNDringSize$: This is the size of the broadcast ring in number of hops. It is set to ρ.

Algorithm 6 NBD Propagation - Algorithm at an Intermediate Node y When Receiving a NBD Message From a Node x.

When node y receives an NBD $(messageID,\ sourceBNID,\ myID,\ BNDringSize,\ numOfHops,\ PATH\_TO\_BN)$ from a node x.
save $PATH\_TO\_BN$ in the routing table as a path to the $sourBNID$ node, which is now a part of the backbone
$numOfHops = numOfHops + 1$
if $(numOfHops \leq ringSize)$ **then**
  /* Add $myID$ to the discovered $PATH\_TO\_BN$ list. */
  $PATH\_TO\_BN = PATH\_TO\_BN \mid myID$
  Broadcast NBD $(messageID,\ sourceBNID,\ myID,\ BNDringSize, numOfHops, PATH\_TO\_BN)$ message to all neighbors
else
  /* Ring size is exceeded. */
  Drop NBD message
end if

- $numOfHops$: This variable contains the number of hops that this message has traversed so far. This variable starts at 0 and is incremented as the BND message propagates through the nodes.
- $PATH\_to\_BN$: This is the path to the BN node that is discovered so far. As the BND message is propagated, each intermediate node concatenates its own ID to the end of the $PATH\_to\_BN$ it received in from the previous node.

Algorithm 6 describes the steps taken by an intermediate node y when it receive the NBD message from another node x. Namely, when the NBD message reaches a node, it does the following. It caches the path, $PATH\_TO\_BN$, to the newly discovered BN node. Node y now can use this path to send messages to the BN node in order to transmit them to the sink through the backbone. It then increments the number of hops. If the new number of hops in the message is still less than or equal to the $ringSize$, then it adds its own ID to $PATH\_TO\_BN$, and broadcasts the message to its neighbors. Otherwise, it drops the message. Figure 2 provides an illustration of the NBD message propagation. As the SF message propagates back from the sink, each of the newly discovered BN nodes broadcasts an NBD message, which is initiated and propagated according to Algorithms 5 and 6 respectively. The figure shows the BN nodes, which are nodes A, B, C, I, K, L, and M. These nodes constitute the discovered backbone. It also shows the NB nodes, which were not designated as part of the backbone. Each of the NB nodes is shown with the corresponding distance (in number of hops) from the nearest BN node. The dashed lines show the path of each of the NB nodes to the nearest BN node according the the described algorithms. These paths are discovered after the broadcast and propagation of the NBD messages from the BN nodes.

III. Performance Evaluation

A. Simulation Setup

This section evaluates the performance of the proposed topology discovery framework, providing insights into the impact of various parameters.

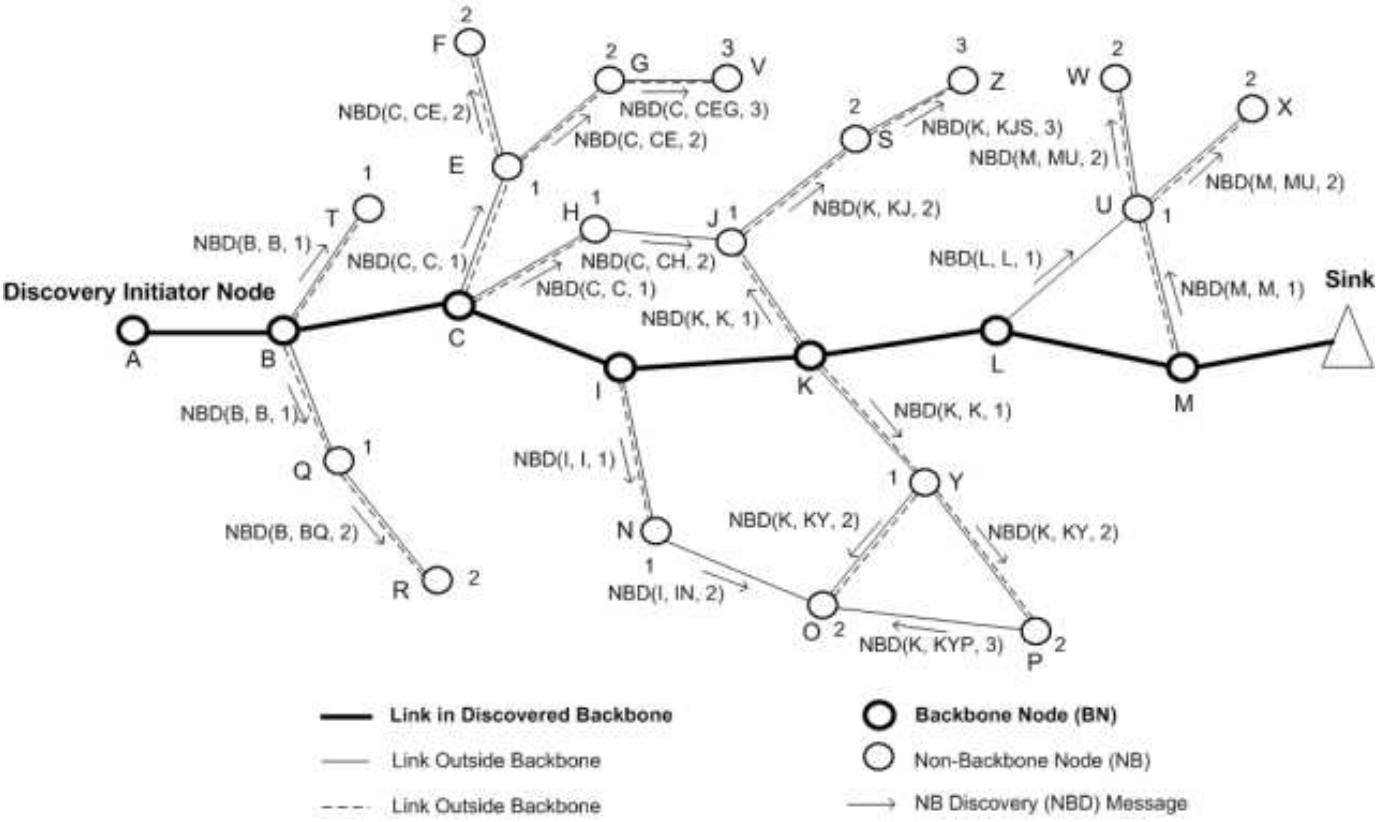
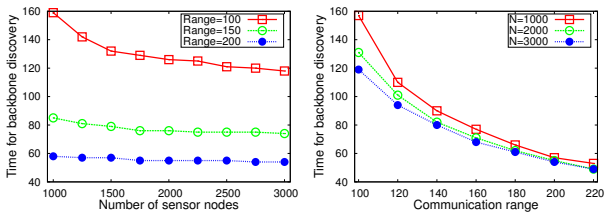
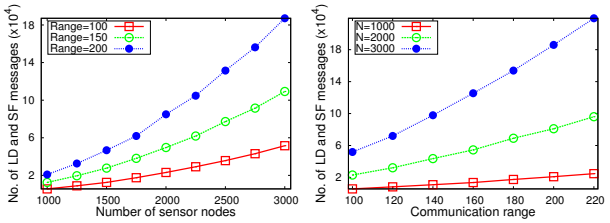


Fig. 2: Illustration of the NBD message propagation in the nearest BN node discovery algorithm.



(a) Varying number of sensor nodes (b) Varying communication range

Fig. 3: Time for backbone discovery



(a) Varying number of sensor nodes (b) Varying communication range

Fig. 4: Number of LD+SF messages

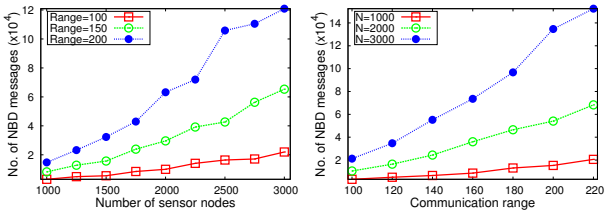
The thick linear sensor network is generated according to the model stated in Sections I and II. A thick LSN is modeled as a rectangle in our simulations. Key parameters in the simulations include the thickness (i.e. the width) of the thick LSN, the length of the thick LSN, the number of sensor nodes, the communication range of a sensor node, and the size of the broadcast ring ρ. In our simulation, the default values of these input parameters are set as follows: the width W is 500 meters, the length L is 10000 meters, the number N of sensor nodes is 1000, the communication range $Range$ of each sensor node is 100 meters, the default ring size ρ is $\frac{W}{2Range} - 1$, which is equal to 2.

In all simulations, the position of each sensor node is uniformly generated within the 2-dimensional rectangle that represents the thick LSN. Two sensor nodes can communicate if and only if the distance between them is not larger than the communication range. The node that initiates the backbone discovery is the leftmost node within the 2-D rectangle; similarly, the sink is the rightmost node within the thick LSN.

The performance metrics used in our evaluations are the time for backbone discovery, the number of LD and SF messages used in the backbone discovery process, and the number of new backbone node declaration (NBD) messages. Our simulation seeks to investigate the impacts of these parameters. Thus, we ran experiments with one varying parameter while keeping the others to their default values. Each experiment run lasts for sufficiently long time, so as to better reflect the performance of the proposed algorithm.

B. Simulation Results

Fig. 3 shows the time for topology discovery while varying the number of sensor nodes and varying the communication



(a) Varying number of sensor nodes (b) Varying communication range

Fig. 5: Number of NBD messages

range. We make two important observations. Firstly, we find that, if the communication range is fixed, when the number of sensor nodes increases, the time for backbone discovery decreases in general. This is because, when the number of sensor nodes increases, since the size of the thick LSN is fixed, the density of sensor nodes increases as well. Consequently, the number of communication links between sensor nodes increases, which finally translates into the decreased number of hops between the initiator and the sink. Secondly, for a large communication range, the time for backbone discovery decreases more slowly. For example, if the communication range is 100, when the number of sensor nodes increases from 1000 to 3000, the time for backbone discovery decreases by 25%; however, when the communication range is 200, the time for backbone discovery decreases by 7%. The reason is that, when the communication range is large, the number of possible communications links is large too, which mitigates the positive effect from the increase of the number of sensor nodes.

Fig. 4 demonstrates the results of the impact of the number of sensor nodes and communication range on the number of LD and SF messages. Generally, if the communication range is fixed, when the number of sensor nodes increases, the number of LD and SF messages increases; similarly, if the number of sensor nodes is fixed, when the communication range increases, the number of LD and SF messages increases too. LD messages are used by the sensor nodes, especially the sensor node that initiates the backbone discovery process, to find the shortest path to the sink node; the SF messages are used by the sensor nodes, especially the sink node, to notify the backbone nodes that they are part of the discovered backbone. Given the meanings of these two types of messages, it is not hard to see the trend in the figures.

We also plot the impact of varying number of sensor nodes and varying communication on the number of NBD messages. The simulation results are shown in Fig. 5. The new backbone node declaration messages are used by the backbone nodes to inform all of the sensor nodes that they are part of the newly constructed backbone. Part (a) of Fig. 5 shows that, if the communication range is fixed, when the number of sensor nodes increases, the number of NBD messages increases too. This is because, when the number of sensor nodes increases, there are more backbone and non-backbone nodes within the thick LSN, thus, the new backbone declaration messages that are broadcast by a backbone node have to be broadcast to a larger number of nodes. On the other hand, part (b) of Fig. 5 shows that, if the number of sensor nodes is fixed, when the communication range increases, the number of NBD messages increases as well. The main reason behind this phenomenon is that, a large communication range enables every sensor to directly communication with a larger number of other sensor nodes, therefore, the number of new backbone declaration messages increases.

In summary, the proposed topology discovery framework works well in a variety of settings in thick LSNs. We hope our findings can reveal some potential insights for future related research.

IV. Conclusions and Future Research

Due to the linear nature of the structure or geographic area that is being monitored, the topology of some WSNs exhibits a linear form. The resulting network was defined in our previous work as an LSN. In this paper, we present a graph-search-based algorithm for backbone discovery in thick LSNs. The resulting backbone can be used for efficient routing of the data collected by the other nodes in the the network. The routing strategy can then take advantage of the linearity of the network and discovered backbone in order to enhance the robustness and fault tolerance of the network. Our future work, will focus on the these important issues to use the linearity of the backbone to allow the routing protocol to overcome node failures by jumping over failed nodes, or going around them through local backbone maintenance.

References

[1] I. Jawhar, N. Mohamed, and D. P. Agrawal. Linear wireless sensor networks: Classification and applications. *Elsevier Journal of Network and Computer Applications (JNCA)*, 34:1671–1682, 2011.

[2] S. Diggavi, M. Grossglauser, and D. Tse. Even one-demensional mobility increases adhoc wireless capacity. *IEEE Transactions on Information Theory*, 51(11), November 2005.

[3] A. Ghasemi and S. Nader-Esfahani. Supporting aggregate queries over ad-hoc sensor networks. *IEEE Communications Letters*, 10(4):251–253, April 2006.

[4] D. Miorandi and E. Altman. Connectivity in one-dimensional ad hoc networks: a queuing theoretical approach. *Wireless Networks*, 12(5):573–587, September 2006.

[5] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys (CSUR)*, 37:164–194, March 2005.

[6] R. Ramanathan and R. Rosales-Hain. Topology control of multihop wireless networks using transmit power adjustment. *In Proc. IEEE Infocom 2000*, pages 404–413, March 2000.

[7] M. Hajiaghayi, N. Immorlica, and V. Mirrokni. Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks. *in: Proc. MobiCom*, September 2003.

[8] Badri Nath B. Deb, S. Bhatnagar. A topology discovery algorithm for sensor networks with applications to network management. *IEEE CAS workshop*, September 2002.

[9] Y. Wang. Topology control for wireless sensor networks. *Chapter 5, Wireless Sensor Networks and Applications, Springer*, pages 113–140, 2008.

[10] A. D'Costa, V. Ramachandran, and A. M. Sayeed. Distributed classification of gaussian space-time sources in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1026–1036, August 2004.