Efficient Topology Discovery and Routing in Thick Wireless Linear Sensor Networks

May 1, 2017

Imad Jawhar[1], Sheng Zhang[2], Jie Wu[3], Nader Mohamed[4], and Mohammad M. Masud[5]

[1]Midcomp Research Center, Saida, Lebanon

[2]State Key Laboratory for Novel Software Technology, Nanjing University, P. R. China

[3]Dept. of Comp. and Inf. Sciences, Temple University, Philadelphia, PA, USA

[4]Middleware Technologies Labs, Isa Town, Bahrain

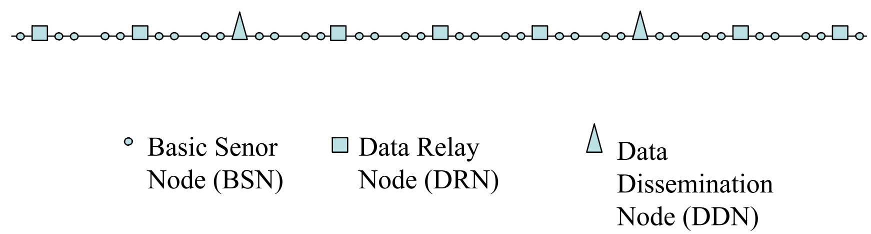
[5]College of Information Technology, United Arab Emirates University, Al Ain, UAE

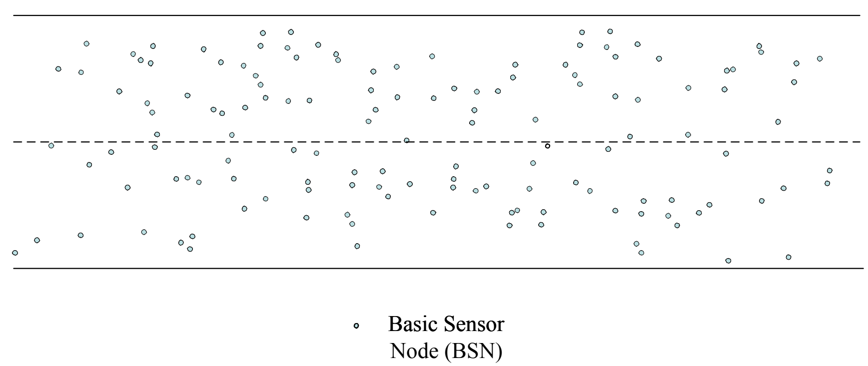
Outline

- **Introduction**: Linear Sensor Networks (**LSNs**). **Applications** and **architectures**
- **Thick LSN model** and **definitions**
- **Algorithms** for **backbone discovery** in thick LSNs
- **Simulation** and results
- **Conclusions** and future research

Linear Sensor Networks (LSNs)

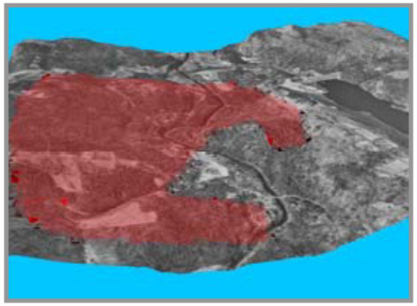
- Wireless sensor networks (**WSN**) advancements in technology
- Sensor networks **application**: environmental, military, agriculture, inventory control, healthcare, etc.
- **Existing WSN** research is 2-D or 3-D deployment.
- Assumption that the network used for sensors does **not** have a **predetermined structure**.
- **Linear alignment** of sensors can arise in many **applications**
- **Linear** characteristic can be utilized for **enhancing** the **routing** and **reliability** in the such systems.
- We can design **adapted protocols** for this special kind of sensor networks.

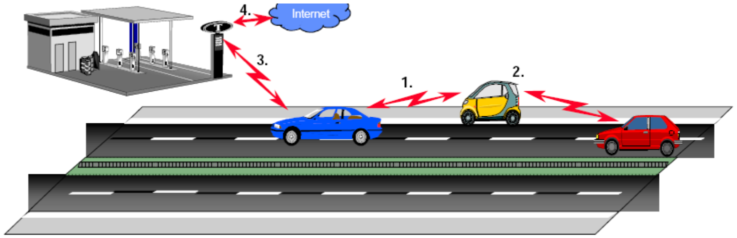




Applications of LSNs

- Oil, Gas, and Water Pipeline Monitoring
- Border Monitoring
- IVC Network
- Railroad/subway monitoring
- Other applications: River, and sea cost monitoring, etc.





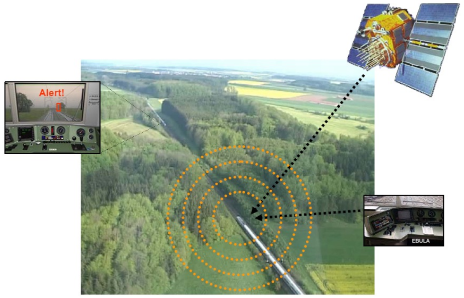
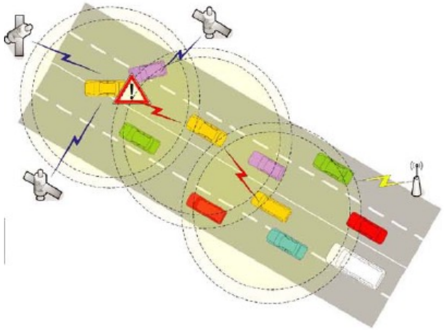
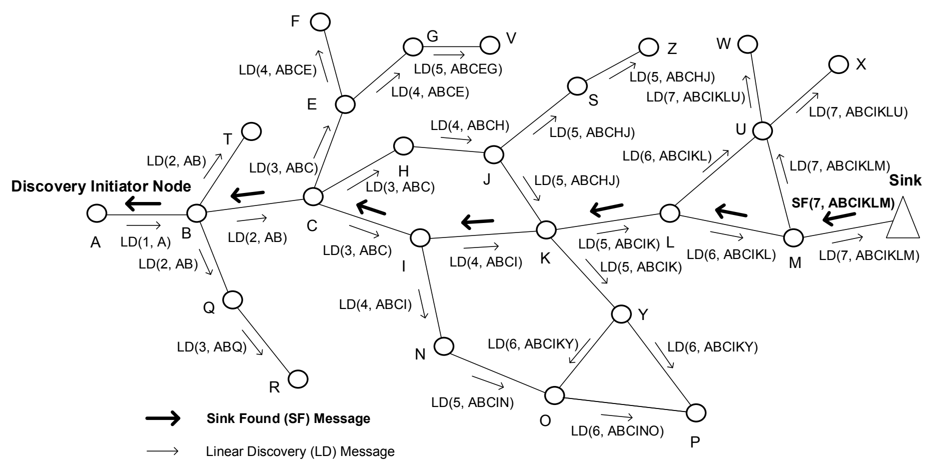


Fig. 1: Interaction of the components of RCAS



Graph-Search-Based Topology Discovery Algorithm for LSNs

- Nodes **identify nodes** to be included in the **backbone** to reach the sink.
- Backbone discovery **increases** the **efficiency**, and robustness of the network.
- Allows more **scalability** of communication along LSN which can have large number of nodes (hundreds or thousands)
- Can enhance **reliability** by “**jumping**” over failed by increasing communication range.
- **No need** for **location** detection (e.g. **GPS**), with higher cost and complexity of SNs**.**
- **Linear Backbone Discovery (LBD) Algorithm**
- Node at **primary edge** sends Linear Discovery (**LD**) **message.**
  - **Message ID**: to prevent **looping**
  - **myID**: ID of **sending** node
  - **messageLC**: linear discovery **counter**. Current **count** from primary edge node.
  - **PATH**: ordered list of **nodes** contained in discovered path



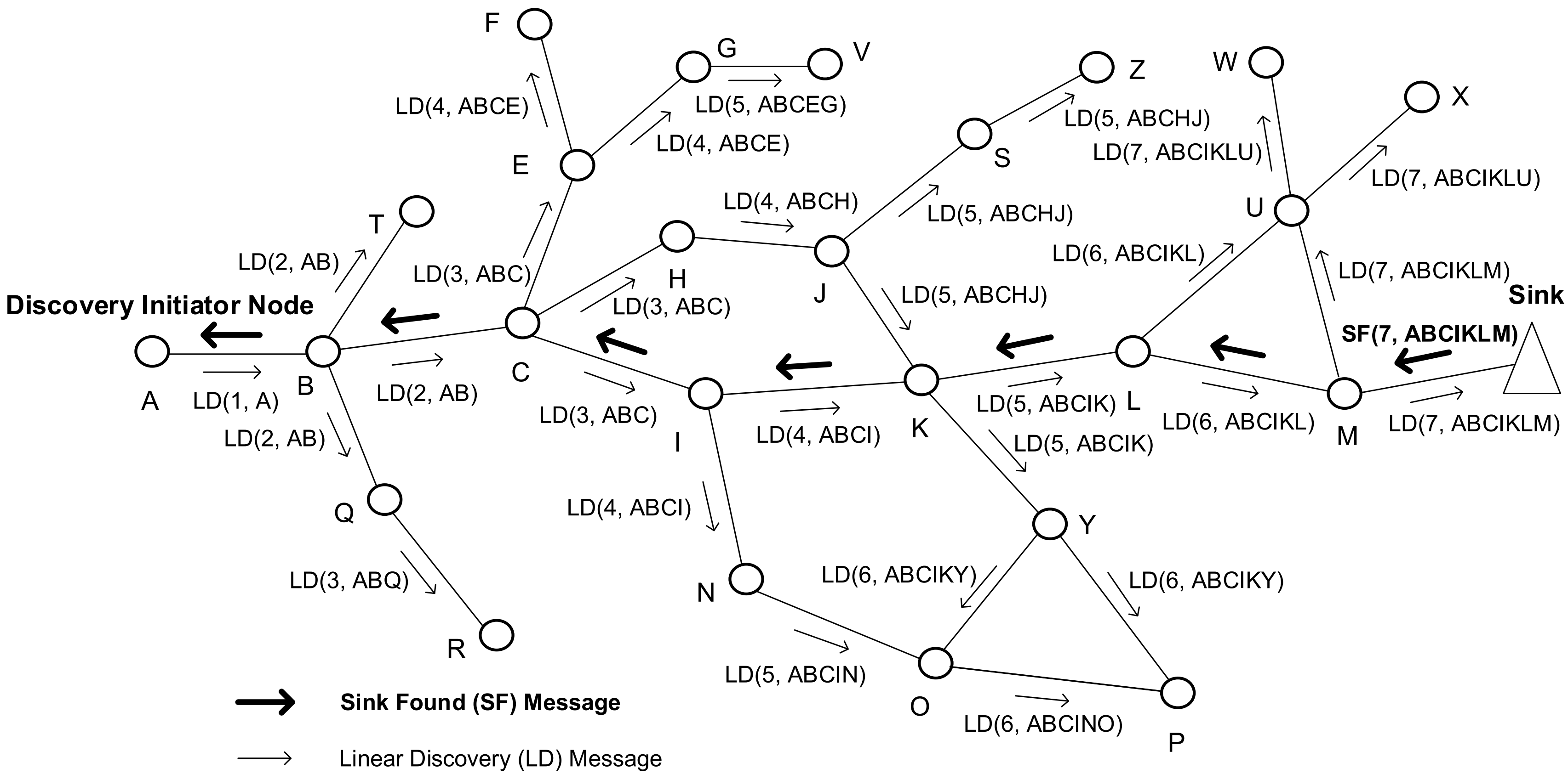
Algorithm 1 Backbone Discovery - Initialization of Node Discovery Variables and Broadcasting of the LD message From the First Node at the Primary Edge

$myColor = WHITE$
/* 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

LD Message Propagation – LBD Algorithm



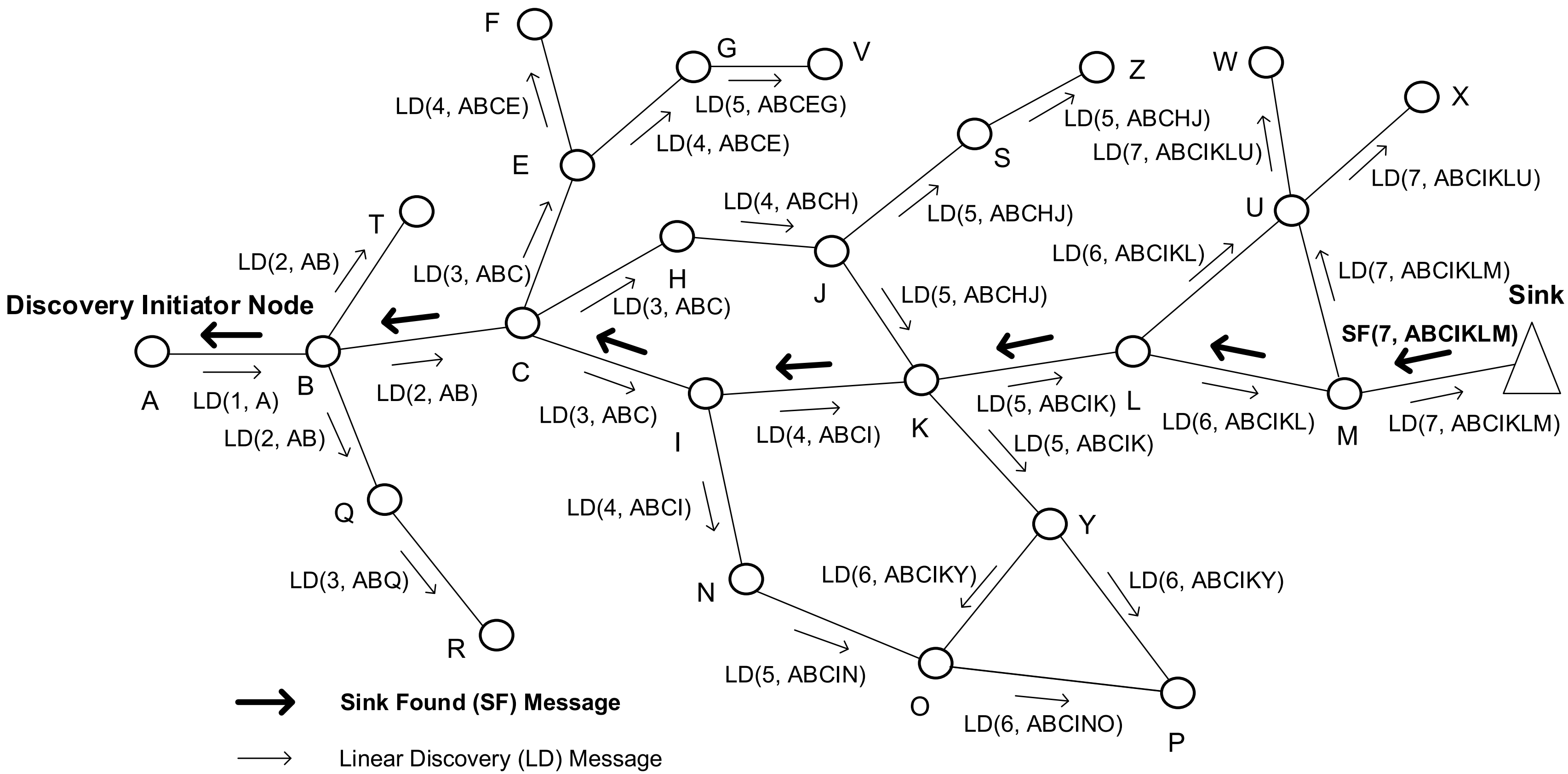
Algorithm 3 Backbone Discovery - Algorithm at the Sink when Receiving a Linear Discovery LD Message From a Node x

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 $SF(messageID,\ source = myID,\ destination = x,\ BBlc,\ PATH)$

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

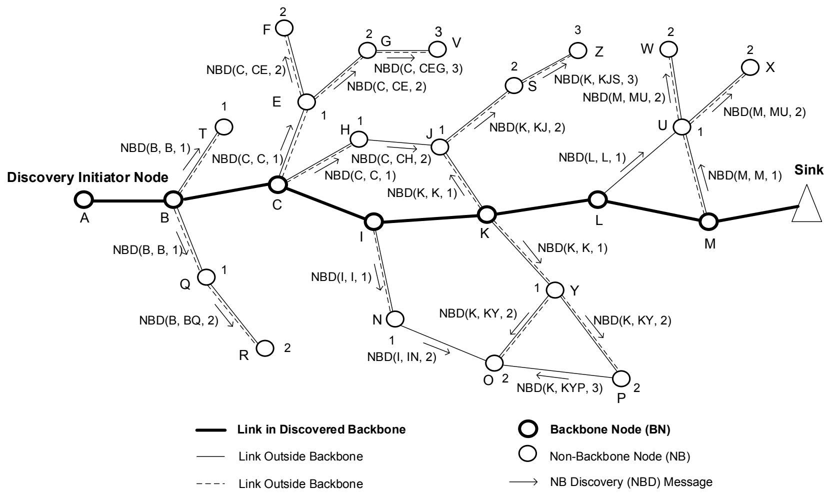
When node y receives the $SF(messageID,\ x,\ messageLc,\ PATH)$ message from node x it does the following:
/* Confirm being a part of the discovered backbone, and cache the discovered backbone in $PATH$ in the routing table. */
$iAmPartOfBackbone$ = $TRUE$
Save the full or local part of the discovered 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 $SF(messageID,\ source = myID,\ destination = x,\ messageLc,\ PATH\ )$

LD Message Propagation – LBD Algorithm



The New BN Declaration (NBD) Algorithm - Initialization

- **Two types of nodes**:
  - **Backbone Nodes** (**BNs**): part of the backbone. Can be used for **routing**, and other functions (data compression, etc.)
  - **Non-Backbone Nodes** (**NBs**): not part of backbone. Can perform **basic sensing** operation.
- **NB** nodes need to find **paths** to nearest **BN** nodes to use them for routing.
- The **newly discovered BN** nodes will **broadcast** a **New BN Declaration** (**NBD**) **message** to accomplish this task.
- **NBD message** has the following fields:
  - **messageID**: to prevent looping
  - **sourceBNID**: ID of **BN** node
  - **myID**: ID of **forwarding** node
  - **BNDRingSize**: size of broadcast **ring** ρ
  - **numOfHops**: traversed number of hops **from BN node**
  - **PATH_to_BN**: accumulated **path** to BN node



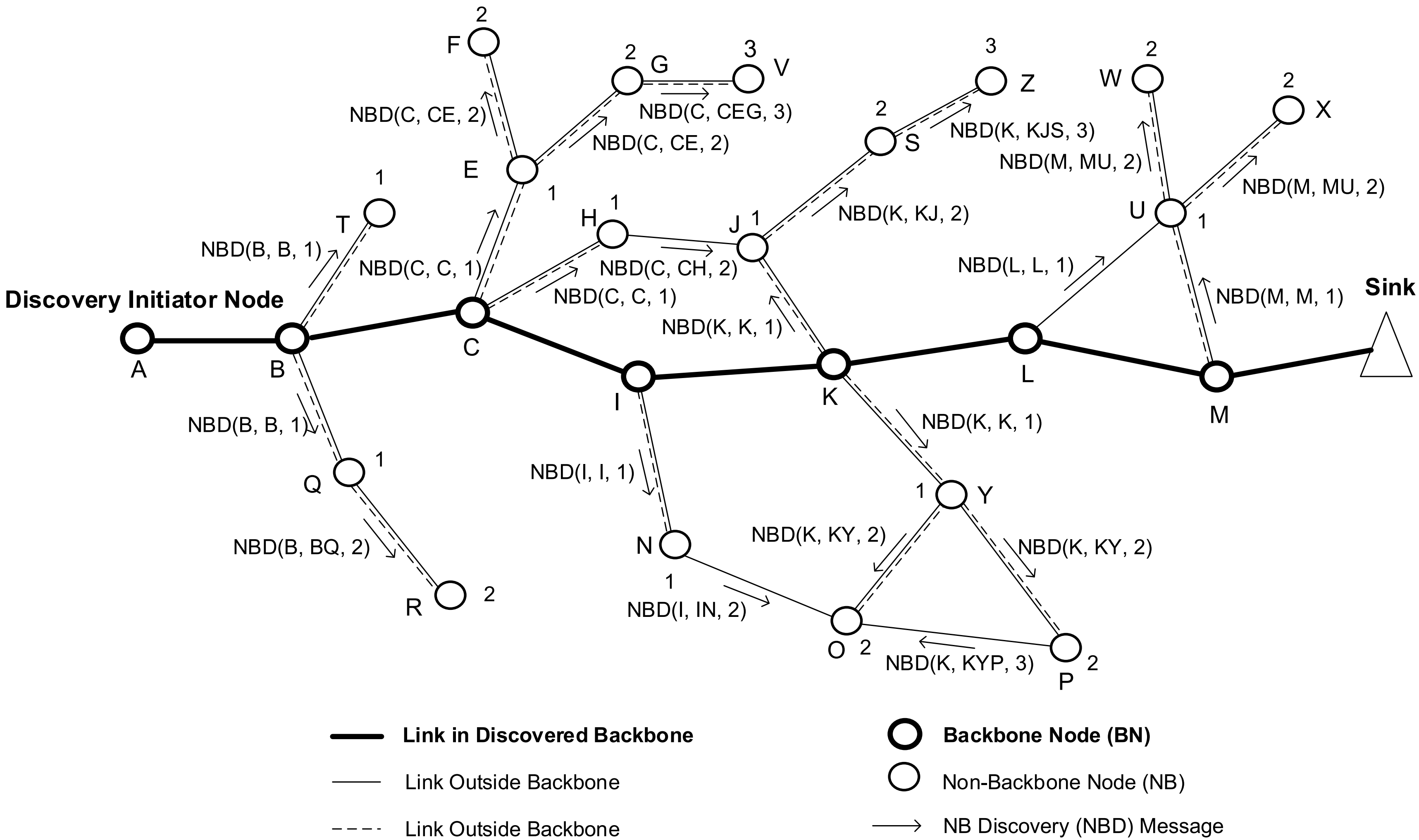
Algorithm 5 NBD Initiation - Algorithm initiated by a newly declared BN node

/* 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 ($messageID$, $sourceBNID$, $myID$, $NBDringSize$, $numOfHops$, $PATH\_TO\_BN$)

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

NBD Message Propagation in New BN Node Discovery Algorithm



The LNBN and L2BN Algorithms

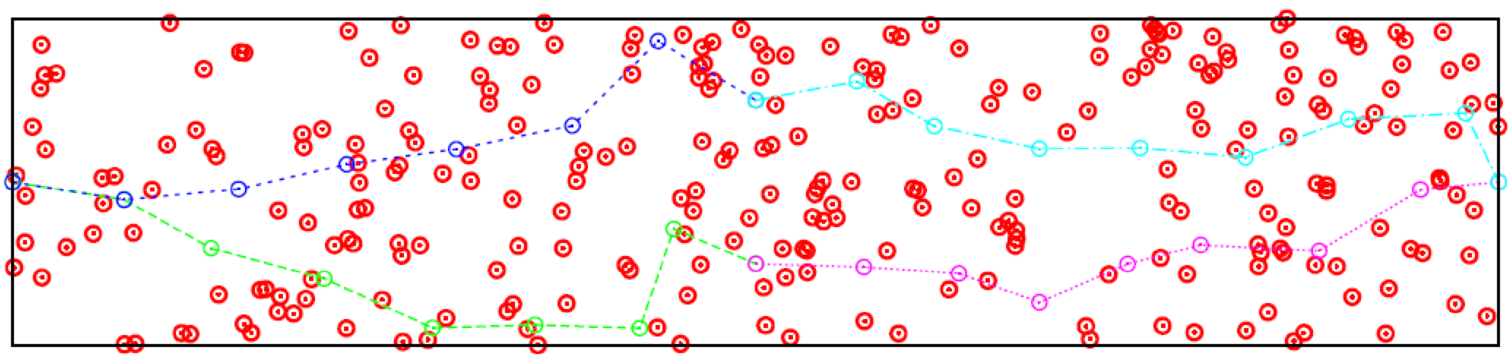
- **Two metrics**
  - Number of **generated messages** for discovery
  - Average **number of hops** for each SN to send messages to the sink
- **LNBN**: does **not** explicitly **minimize** the number of **hops** to the sink
- **Flooding** can be used to **minimize** the number of **hops**. **Each SN** send **LD** LD message to **sink**. **Extreme case**
- **L2BN balances** the two strategies.
- Discover **backbone** with two **paths** using **anchor** nodes in the **middle**.
- Consider **thick LSN** with:
  - L: length
  - T: thickness
- Requires four anchor nodes
  - I: the **discovery initiator**
  - S: the **sink**
  - **Two** other **anchor** nodes:
    - **U(L/2, T/4)**
    - **V(L/2, 3T/4)**
- With **upper** and **lower paths SNs** in the upper and lower regions have **shorter path** to sink

Algorithm 7 L2BN

Require: I, S, U, V

1: use LBN to find the shortest path I-U between I and U
2: use LBN to find the shortest path U-S between U and S
3: use LBN to find the shortest path I-V between I and V
4: use LBN to find the shortest path V-S between V and S
5: catenate I-U and U-S
6: catenate I-V and V-S
7: use NBN for the other nodes to find their shortest paths to the constructed two backbone paths

Note these two paths are **not necessarily** node **disjoint**



Simulation

- Simulation to **validate** and **evaluate** the algorithms.
- **Thick LSN** generated according to **model**.
- Modeled as **rectangle** in our simulation
- **Key parameters**:
  - **Thickness** of LSN (i.e. width): W – Set to 500 m.
  - **Length** of LSN: L – set to 10000 m.
  - **Number** of sensor nodes: N – Set to 1000
  - Node communication **range**: Range – Set to 100 m.
- **Position** of each sensor node **uniformly generated** within 2-dimensional rectangle
- **Initiator** node is **leftmost** node in 2-D rectangle
- **Sink** is **rightmost** node
- **Performance metrics**:
  - **Time** for backbone discovery
  - **Number** of **LD and SF messages** used in discovery
  - **Number** of new backbone declaration (**NBD**) **messages**

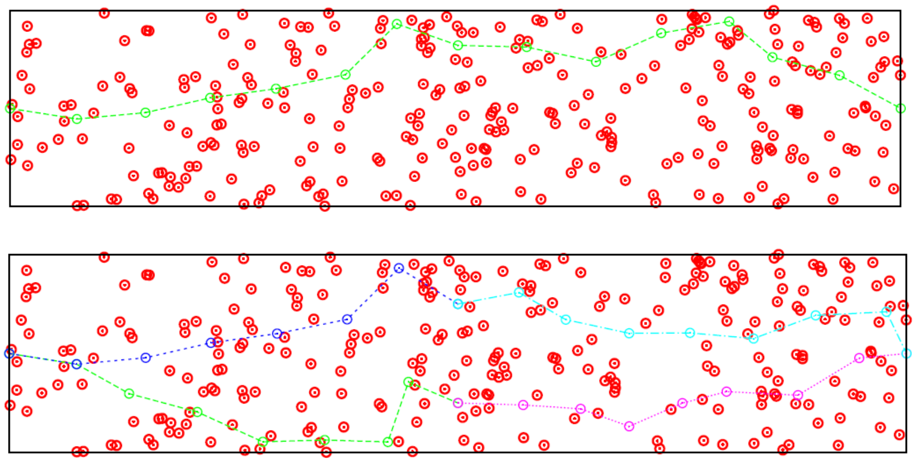
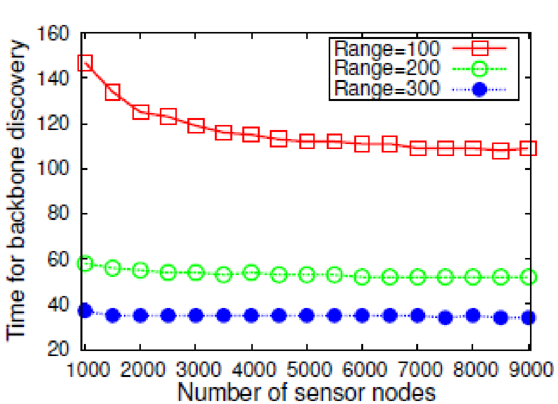
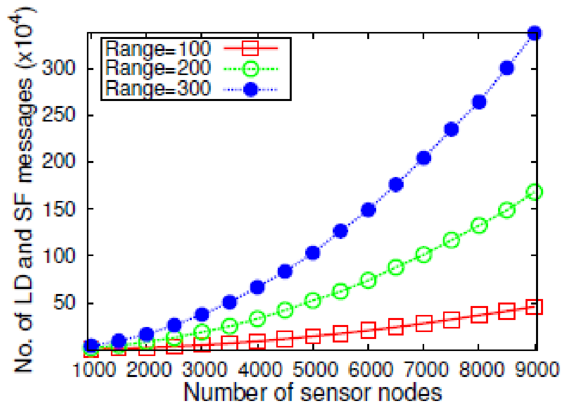


Illustration of the backbone path where W = 500, L = 2500, N = 300, and Range = 200.

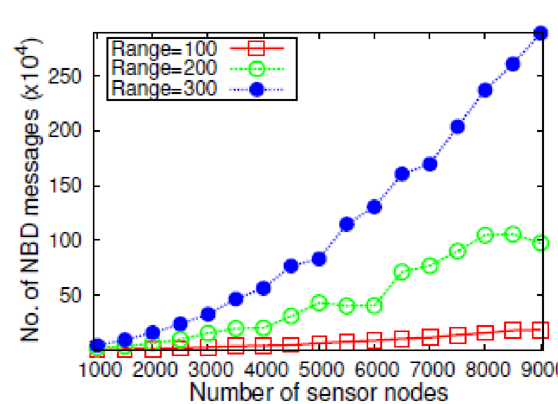
LNBN on large instances



(a) Time of Discovery



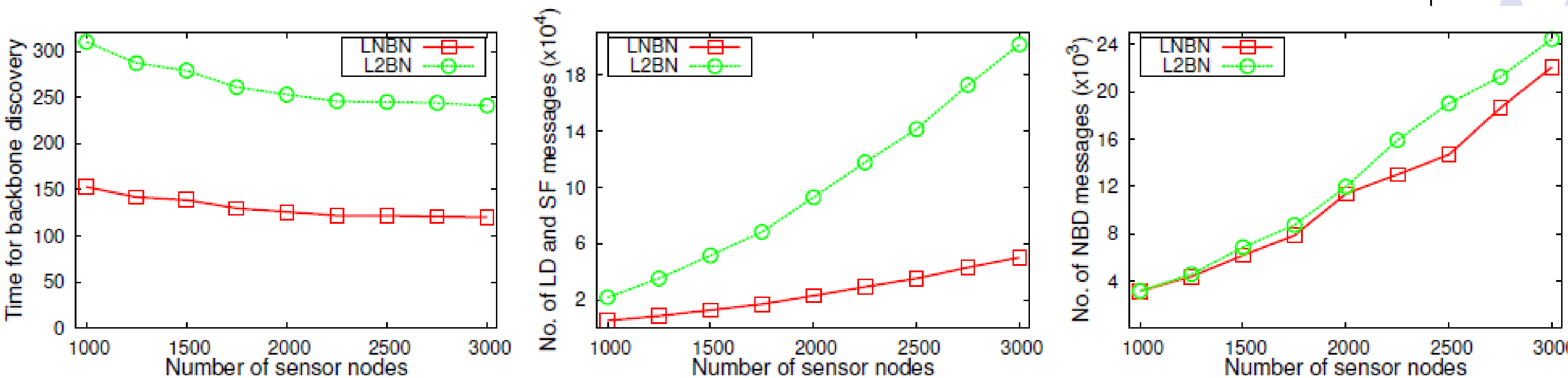
(b) Number of LD+SF messages



(c) Number of NBD messages

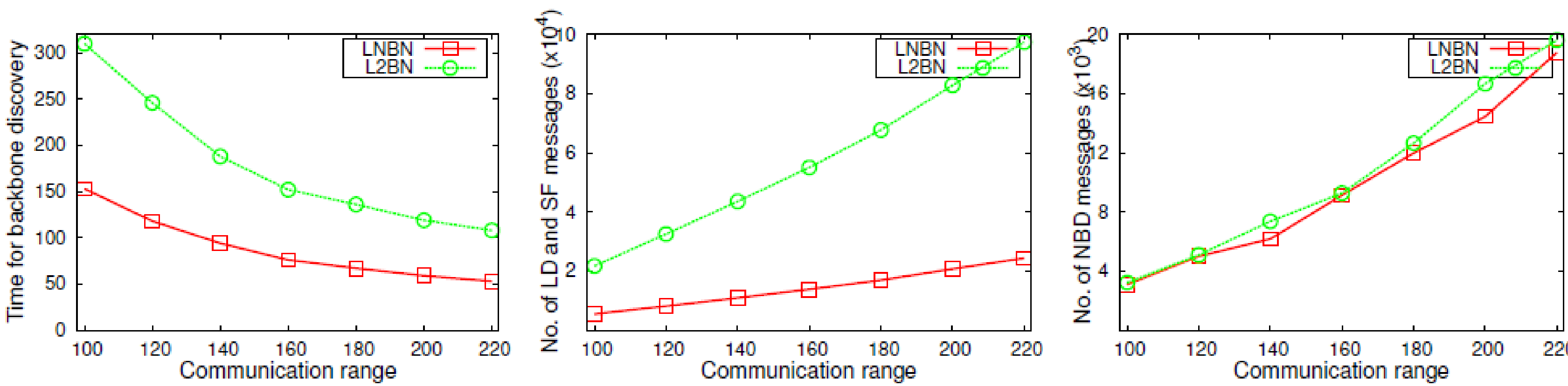
- When number of **SNs increases**, backbone **discovery** time **increases.**
- Number of **LD+SF messages increases** as number of **SNs increases** increasingly, the increasing speed also increases: messages are spread in a **broadcast** nature, so messages increase **proportionally** to the **square** of the number of SNs.

Comparison results of LNBN and L2BN



(a) Time of Discovery (b) Number of LD+SF messages (c) Number of NBD messages

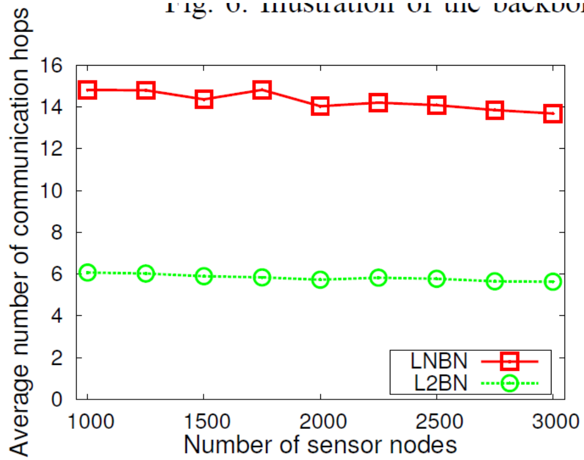
Comparison of **LNBN** and **L2BN** under **varying number** of **nodes** while fixing the range at 100

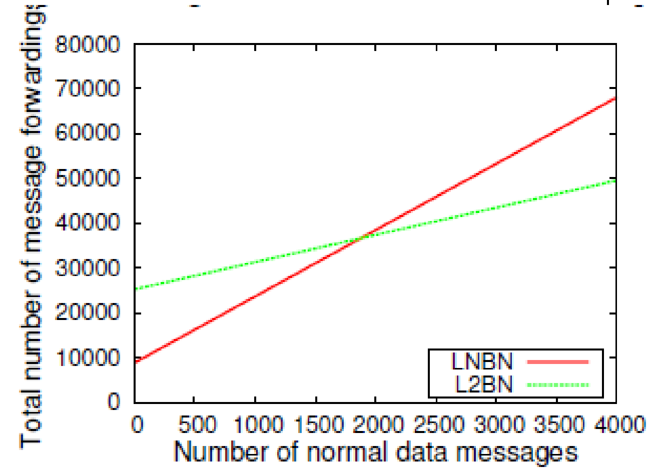


(a) Time of Discovery (b) Number of LD+SF messages (c) Number of NBD messages

Comparison of **LNBN** and **L2BN** under **varying range** while fixing the number of nodes at 1000

Average No. of Hops and Total No. of Message Forwardings





- When number of normal **data messages** exceeds **2,000**, total number of message **forwardings** in **L2BN** becomes **less** than that of LNBN.
- Since number of **SNs** in the WSN of interest **typical** exceeds **1,000**, the number of normal **data messages** can easily **exceed 2,000**.

Conclusions

- Stated some of the **applications** for **thick LSNs** in order to **motivate** the research
- Presented graph search algorithm for **backbone discovery** in thick LSNs. **c**an be **used** for **efficient routing to sink**.
- **Two different algorithms** were presented:
  - **LNBN algorithm** to discover a path from initiator node to the sink on the other end, and then uses **NBD** broadcast **Algorithm** to discover paths between NB and BN nodes.
  - **L2BN algorithm** to discover **two backbone paths** using two **anchor** nodes in the middle of the LSN.
- Algorithms have good **scalability.**
- For **long thick LSNs**, can use **multiple segments** separated by sinks for added **efficiency**, **reliability**, and **scalability.**
- **Thick LSNs** require more research. More **optimizations** to enhance **routing**, **reliability**, and **energy** efficiency such as **jumping** over failed nodes.

Thank you.

Questions?