

A Resource-Efficient and Scalable Wireless Mesh Routing Protocol*

Jianliang Zheng[†] and Myung J. Lee

Department of Electrical Engineering, The City College of New York, New York, NY 10031

Abstract

By binding logic addresses to the network topology, routing can be carried out without going through route discovery. This eliminates the initial route discovery latency, saves storage space otherwise needed for routing table, and reduces the communication overhead and energy consumption. In this paper, an adaptive block addressing (ABA) scheme is first introduced for logic address assignment as well as network auto-configuration purpose. The scheme takes into account the actual network topology and thus is fully topology-adaptive. Then a distributed link state (DLS) scheme is further proposed and put on top of the block addressing scheme to improve the quality of routes, in terms of hop count or other routing cost metrics used, robustness, and load balancing. The network topology reflected in logic addresses is used as a guideline to tell towards which direction (rather than next hop) a packet should be relayed. The next hop is derived from each relaying node's local link state table. The routing scheme, named as topology-guided DLS (TDLS) as a whole, scales well with regard to various performance metrics. The ability of TDLS to provide multiple paths also precludes the need for explicit route repair, which is the most complicated part in many wireless routing protocols. While this paper targets low rate wireless mesh personal area networks (LR-WMPANs), including wireless mesh sensor networks (WMSNs), the TDLS itself is a general scheme and can be applied to other non-mobile wireless mesh networks.

Keywords: Wireless mesh networks; Mesh routing; Scalability; Adaptive block addressing; Distributed link state

1 Introduction

Wireless networks have been infiltrating homes, offices, universities, and other industrial and commercial premises around the globe, thanks to the advances in many wireless technologies. This commercial success has in part stim-

ulated the development of wireless mesh networks. The persistent driving force, however, comes from the envisioned advantages of wireless mesh networks themselves. First of all, mesh connectivity significantly enhances the network performance, such as fault tolerance, load balancing, and throughput; in addition, the self-configuring and self-healing feature of wireless mesh networks not only enables them to be deployed on the fly and on the cheap, but also enhances system resilience and reliability; moreover, the ability of wireless mesh networks to provide flexible coverage areas, particularly where other wireless technologies tumble due to the lack of line of sight (LOS), makes them unique for many applications. With all these, plus the minimal up-front investment and the simple off-the-shelf devices (as opposed to sophisticated devices used for central control purpose in other networks) that are outfitted with radio communications gear, it is conceivable that wireless mesh networks will be one of most competitive wireless communication solutions.

Nevertheless, before the promise of wireless mesh networks can be realized, much research remains to be done [5]. In this paper, we propose a resource-efficient and scalable wireless mesh routing protocol, called topology-guided distributed link state (TDLS), for low rate wireless mesh personal area networks (LR-WMPANs), which are at the low end on the continuum of wireless mesh networks in terms of data rate and are normally non-mobile. The IEEE 802.15.4 standard [1] specifies the medium access control (MAC) sublayer and the physical (PHY) layer for low rate wireless personal area networks (LR-WPANs), and ZigBee Alliance [2] has been developing other upper layers. LR-WPANs, as an enabling technology, possess some unique design features and show promise to bring ubiquitous networking into our lives [6]. Routing protocols designed for wireless mobile ad hoc networks (MANETs) [8–12] can be directly applied to some mesh networks. Standard Internet routing protocols [13, 14] can also be used if the network topology is relatively stable. However, those routing protocols may be too cumbersome for LR-WMPANs, given the constraints on resources of those networks. TDLS, on the other hand, is a light-weight scalable routing protocol that well caters to the requirements of resource-constraint networks such as LR-WMPANs and WMSNs. In TDLS, the network topology reflected in logic addresses assigned

*The research has been supported by the Samsung Advanced Institute of Technology.

[†]Corresponding author. Tel: 1-212-650-7199; Fax: 1-212-650-8249; E-mail: zheng@ee.cuny.edu

during the setup of a network is used as a guideline to tell towards which direction (rather than next hop) a packet should be relayed. The next hop is derived from each relaying node’s local link state table.

Although at high level TDLS is merely an approach that combines topology information and link state, it owns distinct features and is different from existing geographical routings [15, 16], link state routings [11, 17], or hierarchical routings_ [18–22]. TDLS uses logical topology information (i.e., tree structure), which is a byproduct of network auto-configuration. On the other hand, geographical routings use geographical topology information, which is often obtained through the Global Position System (GPS). GPS is far too expensive for resource-constrained low-cost LR-WMPANs and WMSNs. TDLS is a fully distributed scalable scheme. For each node, the control overhead is that it needs to broadcast several Hello messages to its neighbors within n (e.g., 3) hops during the network auto-configuration phase. And a node only needs to keep several hundred bytes of link state information for those neighbors, regardless of the network size. However, for most existing link state routings, the resource consumption (e.g., power and memory) on each node will increase as the network size increases. Those link state routings do not scale well in LR-WMPANs and WMSNs, which may contain large number of tiny devices that are battery-powered and memory-constrained. A plethora of wireless routing protocols exploit hierarchical routing structure to achieve scalability. For example, Zone Routing Protocol (ZRP) [18] performs routing at two different levels, i.e., proactive intra-zone routing and reactive inter-zone routing. Unless the source and destination of a packet belong to the same routing zone, the packet needs to be first forwarded to the zone where the destination sits using reactive routing, and then be routed to the final destination using proactive routing. As the network size increases, either the number of routing zones or the radius of each routing zone needs to be increased, which may still result in scalability problem. By contrast, TDLS uses a simple flat routing structure to solve the scalability problem. In TDLS, each node maintains its n -hop link state, which is not affected by the network size. So TDLS is fully distributed and scalable in terms of control overhead and memory consumption. Since the logic topology information tells toward which direction a packet should be relayed, no reactive routing is needed as in ZRP. No matter how far away the destination is, relaying decision is always based on the relaying node’s local link state. This also means TDLS has a better support for multiple paths and load-balancing than ZRP, because ZRP can only support multiple paths and load-balancing during its intra-zone routing period. By providing multiple paths, TDLS precludes the need for explicit route repair, which is the most complicated part in many wireless routing protocols.

The remainder of this paper is organized as follows. An adaptive block addressing scheme is first introduced in section 2. Besides supporting auto-configuration and performing logic address assignment, it is also a self-contained routing protocol. Then in section 3, details of the distributed link state scheme are covered, including the basic link state scheme, the extended link state scheme, link state generation, data forwarding, sanity/consistency checking, and link state maintenance. Next, simulation results are presented and analyzed in section 4. Finally, in section 5, we conclude the paper with a summary.

2 An Adaptive Block Addressing Scheme

The adaptive block addressing scheme presented here is used for logic address assignment as well as network autoconfiguration. It is also a self-contained routing protocol and can perform either the so-called adaptive tree (AT) routing or the meshed AT (MAT) routing. The details of AT and MAT will be covered in subsection 2.2 and 2.3 respectively.

2.1 Background

A logic tree structure can be formed along with the setup of a wireless network. The first node in the network will designate itself as the root and begin to accept association requests from other nodes. Any node already in the network can determine whether to allow other nodes to join it, that is, whether to act as a router, depending on the availability of its resources such as memory and energy. Various tree algorithms have been proposed for routing purpose in both wired and wireless networks [23–29]. Of our particular interest here is the cluster tree algorithm [23], which has been adopted by ZigBee [3]. Two network-wide parameters are defined in cluster tree: the maximum number of children a node can have, C_m , and the maximum level of the tree (i.e., the maximum hops from the root to a leaf node), L_m . A cluster tree can be formed through a top-down procedure. First, the root distributes the whole network address space evenly among its maximum allowed C_m children¹; then each of the children further distributes the allocated address block evenly among its C_m children; and this procedure continues until the maximum level, L_m , is reached. The address allocation is solely based on C_m and L_m , irrespective of the number of children a node actually has. By knowing C_m , L_m , and its own tree level, a node is able to calculate the next hop by looking at the destination address in a packet.

¹Each child still gets $1/C_m$ of the total available addresses even if the actual number of children is less than C_m .

Cluster tree, however, lacks flexibility and robustness. The inflexible address assignment often results in huge waste of addresses (our simulation shows that a network with a 16-bit address space can only accommodate several hundred nodes). Cluster tree also suffers for single point of failure (SPF). Among other problems are non-optimal routes and the need for address reassignment when a node changes its attaching point.

To solve the aforementioned cluster tree problems, we propose a new type of tree called adaptive tree (AT), which has the following features:

- Addresses are assigned adaptively according to the network topology. This means asymmetric topology will lead to asymmetric address assignment. Different nodes can have different numbers of children, and different branches below a node can have different address block sizes.
- The concept of C_m and L_m no longer exists. A node can accept as many children as it wishes, only if its physical capacity allows. And a tree can have as many levels as it needs. The limitation only comes from the address space.
- If needed, a meshed AT (MAT) can be formed. By using a MAT, it is possible to route a packet through a shorter path. Some SPFs can also be removed by using a MAT.

2.2 Adaptive Tree (AT)

2.2.1 Two Phases of AT

Functionally, two phases are defined in AT: initialization (or configuration) phase and operation phase.

- During the initialization phase, nodes join the network and an AT is formed.
- After initialization, the network enters the operation phase and normal data communications start. During the operation phase, new nodes are still allowed to join the network.

2.2.1.1 Initialization Phase

An AT is formed during the initialization phase. The AT formation can be broken down into two stages: association and address assigning.

During the association stage, beginning from the root, nodes gradually join the network and a tree is formed. But this tree is not an AT yet, since no node has been assigned an address. There is no limitation on the number of children a node can accept. A node can determine by itself how many nodes (therefore, how many branches below it) it will accept according to its capability and other factors.

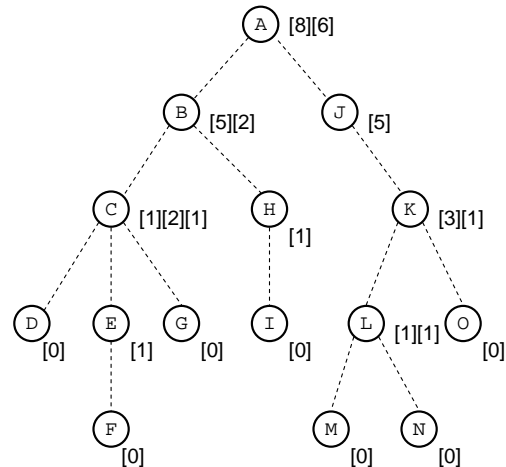


Figure 1: Calculation of Number of Nodes along Each Branch

It is possible that a joining node cannot find any other node to associate if the number of children a node can have is not flexible. Therefore, instead of accepting or rejecting an association request, a node uses an acceptance degree (AD) to indicate the willingness of acceptance. For example, a four-level AD scheme could be:

- 3 – accept without reservation
- 2 – accept with reservation
- 1 – accept with reluctance
- 0 – reject (a node should try to avoid this AD unless absolutely necessary)

When a joining node receives multiple association responses, it should choose the node with the highest AD for association. Using AD increases the chance with which a node successfully joins the network without severely overloading an individual node.

After the tree reaches its bottom, that is, no more nodes are waiting for joining the network (a proper timer can be used for this purpose), a down-top procedure is used to calculate the number of nodes along each branch, as shown in Figure 1. The numbers in brackets indicate the numbers of nodes along branches below a certain node.

When numbers of nodes are reported from bottom to top, each node can also indicate a desirable number of addresses. For example, node *F* can indicate it wishes to get 3 addresses (2 for possible future extension), though currently only one address is enough (for itself); node *E* can indicate it wishes to get 5 addresses (3 for node *F*, 1 for itself, and 1 for future extension); and so on. After the root, node *A* here, receives the information from all the branches, it begins to assign addresses.

During the address assigning stage, a top-down procedure is used. First, the root checks if the total number of

nodes in the network is less than the total number of addresses available. If not, address assignment fails. One possible solution in case of address overflow is to separate the network into smaller ones or augment the address space. Here we do not handle address overflow, and will assume no address overflow happens hereafter. Next, the root assigns a block of consecutive addresses to each branch below it, taking into account the actual number of nodes and the wished number of addresses. The actual number of addresses assigned could be less or more than the wished one, depending on the availability of addresses, but no less than the actual number of nodes. This procedure continues until the bottom of the tree is reached. After address assigning, an AT is formed and each node has an AT table (ATT) for tracking its branches. For example, node *C* could have an ATT as follows:

```
[6][8]
[9][13]
[14][14]
```

The above ATT indicates that node *C* has total 3 branches. Branch 1 owns address block [6, 7, 8]; branch 2 owns address block [9, 10, 11, 12, 13]; branch 3 only owns one address 14.

2.2.1.2 Operation Phase

After an AT is built, the network enters the operation phase. Using the ATT, a packet can be easily routed. As an example, when node *C* receives or generates a packet, it checks if the destination² belongs to one of its branches and, if yes, the packet will be relayed to that branch. If the destination falls out of any branch below node *C*, the packet will be routed to the parent of node *C*.

Although no significant change of the node number or network topology is expected during operation phase, more nodes (therefore, more branches) are still allowed to be added at any level of the tree, only if additional addresses (reserved during initialization phase) are available. Address assignment can be locally adjusted within a branch if a node runs out of addresses. For instance, a node can request more addresses from its parent. If the parent does not have enough addresses, it can try to either request additional addresses from its parent or adjust address assignment among its children. If there is a substantial change of the node number or network topology, which can not be handled locally during operation phase, the network is allowed to go through the initialization phase again. In practice, address assignment is controlled by the application profile. For example, the application profile used for light control in ZigBee can specify that more addresses should be reserved for some special

²The logic address of the destination can be determined by service discovery as usual.

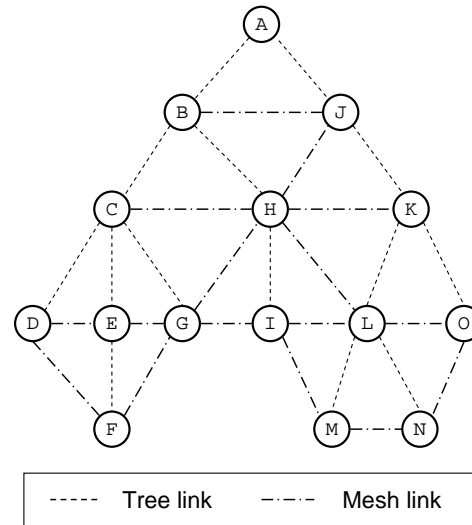


Figure 2: Meshed AT

nodes such as those near hall ways. This further improves the utilization of addresses and reduces the probability of reinitialization as the network evolves.

2.3 Meshed AT (MAT)

A meshed AT (MAT) can be formed on top of an AT. In Figure 2, besides the lines that represent tree links, additional lines (referred to as mesh lines here) are added so that the network now looks more like a mesh than a tree. But from each individual node's point of view, the network is still a tree. Any two nodes connected via a mesh line treat each other as a child and add an ATT entry for each other. For example, node *K* treats node *H* as a child, and vice versa. Notice that ancestors and descendants, no matter they are one level or multiple levels away from the node, are not meshed (i.e., not connected to the node through mesh lines).

By forming a MAT, it is possible to route a packet through a shorter path. For instance, a packet from node *M* to node *I* can be transmitted directly (Figure 2), since from node *M*'s point of view, node *I* is its child. On the other hand, the original path in the corresponding AT is *M-L-K-J-A-B-H-I*. While this is an extreme example, there is a high probability that a shorter path can be found in MAT than in AT. For example, from node *E* to node *H*, the path will be *E-C-H* instead of *E-C-B-H*. Children added due to meshing behavior are likely at different tree levels. For example, nodes *J*, *K*, and *L* are children of node *H*, but at different tree levels. If node *H* has a packet for node *N*, it will find that all the three nodes *J*, *K*, and *L* can relay the packet. In this case, node *H* will choose the node sitting at the largest tree level (that is, farthest from the

root), namely, node *L*. Notice that node *H* can easily find out the relative tree levels of nodes *J*, *K*, and *L* through the address blocks allocated to them.

Another advantage of MAT is that some SPFs are removed. For example, if the link between nodes *J* and *K* is broken, packets from node *K* to node *H* or *I* can still be routed.

3 Distributed Link State

Notwithstanding AT (or its meshed form, MAT) is a self-contained routing protocol, better routing performance can be achieved by combining the block addressing scheme with a scheme we called distributed link state (DLS). DLS is a fully-distributed scheme, as its name tells, which scales well. The additional complexity added by combining the block addressing scheme with the DLS scheme is minimal, in the sense that we merely increase the time to live (TTL) of a Hello message from 1 to *maxHops* (a typical value is 3). Maintaining simplicity is fundamentally critical for the success of WMPANs and WM-SNs that comprise devices with constrained resource like small RAM size. For instance, Atmel [30] 8-bit processor has only a RAM of the order of 10KB, which is even too small to house the AODV routing table in case of several hundreds of nodes. By contrast, TDLS only needs a tiny link state table, e.g., about 300B for 30 neighbors, regardless of the network size. Notice that the above storage saving has not yet accounted for the saving on code size of the routing protocol.

TDLS also maintains many advantages a proactive approach can have, for example, there is no initial route discovery latency and thus is suitable for time critical sensor applications like light control within ZigBee [2]. It is also more efficient than most other proactive approaches, since no periodical update is needed after the link state generation stage (see subsection 3.3 and 3.5). Among other advantages are shorter paths (or better paths when cost metrics other than hop count are used in route selection), multiple paths, and robustness. In what follows, we present the details of TDLS, including the basic link state scheme, the optional extended link state scheme, link state generation, data forwarding, sanity/consistency checking, and link state maintenance.

3.1 The Basic Link State

In the basic link state (BLS) scheme, each node maintains a link state table (LST) for all its neighbors within *maxHops* (a parameter determined by specific applications) hops. Each neighbor's block address is logged in the LST so that the whole branch below it is routable. To forward a data packet, the LST is queried and the best

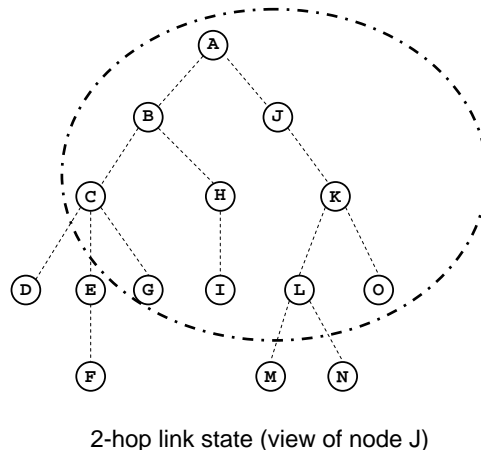


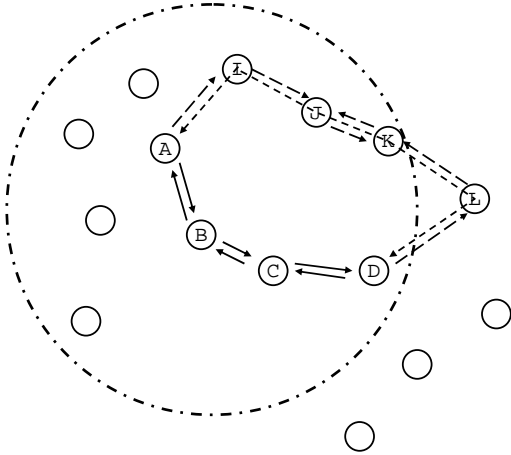
Figure 3: An Example of the Basic Link State Scheme

path (when multiple paths are found) is used (see subsection 3.4 for details). Figure 3 illustrates a 2-hop BLS view of node *J*. In this example, nodes *D*, *E*, *F*, *M*, and *N* are not within 2 hops of node *J*, but they are still directly routable since they are the descendants of those nodes within 2 hops of node *J*.

3.2 The Extended Link State Scheme

The extended link state (ELS) scheme is optional in TDLS. In the ELS scheme, a node is not forced to use the predetermined *maxHops* parameter to build its LST. Although nodes are still required to exchange *maxHops*-hop neighbor information, a node can build an LST for *linkHops*-hop neighbors ($linkHops \leq maxHops$), depending on the node density around it. In general, a node will use a large *linkHops* if the node density is small around it, and vice versa. One of the goals of using DLS is to bypass broken tree routes. With high node density, a node is likely to find more paths that can be used to bypass a broken tree route. This improved reliability enables a node to build a relatively small LST.

If the node density is very low around an area, even a *linkHops* as large as *maxHops* may not be enough. One solution is to allow nodes in this area to use a *linkHops* that is larger than *maxHops*. But since only *maxHops*-hop information is exchanged by default, all nodes in this area need to exchange *linkHops*-hop ($linkHops \geq maxHops$ in this case) information again. Here we propose to use another more efficient approach. After the basic LST is built, each node checks if it has multiple paths to each of the neighbors within *linkHops* hops. If not, it will unicast its complete LST to those neighbors. Upon receiving the LST, a node will unicast back its LST to the source. This in effect creates a $(2 \times linkHops)$ -hop LST between



Multiple paths created through ELS

Figure 4: An Example of the Extended Link State Scheme

related nodes. Figure 4 shows an example of the ELS scheme. After the basic LST is built, node *A* finds that it only has one path to node *D* within *linkHops* (here 3) hops. So it will exchange LST information with node *D*. And both node *A* and node *D* will find out that there is another path between them, that is, *A-I-J-K-L-D*. Then each of them will record this path in the LST and also notify nodes *I*, *J*, *K*, and *L* so that those nodes can also update their LSTs. Finally, a second path is set up between node *A* and node *D*. Among all the neighbors within *linkHops* hops, the ancestors and descendents of a node (according to the tree structure) are more important than other nodes. So optionally a node can handle its neighbors differently, e.g., only guarantees that multiple paths are available to ancestors and descendents, or keeps more hop link state of ancestors and descendents than that of other nodes.

In summary, the ELS scheme is more reliable, efficient and adaptive than the BLS scheme, at the additional cost of complexity.

3.3 Link State Generation

A node should broadcast several Hello messages with a TTL of *maxHops* to exchange link state information with its neighbors when it receives a logic address from its parent, or when it receives a Hello message from a new neighbor and if it has been assigned a logic address. The Hello message format is given in Table 1.

The LST of a node, which consists of a *maxHops*-hop neighbor list (see subsection 3.3.1) and a connectivity matrix (see subsection 3.3.2), is updated upon the reception of each Hello message.

3.3.1 Neighbor List

Table 2: Neighbor List

$begAddr_1$	$endAddr_1$	$tree_level_1$	$hops_1$
$begAddr_2$	$endAddr_2$	$tree_level_2$	$hops_2$
...
$begAddr_n$	$endAddr_n$	$tree_level_n$	$hops_n$
$begAddr_i$:	beginning address of the address block owned by neighbor <i>i</i>		
$endAddr_i$:	ending address of the address block owned by neighbor <i>i</i>		
$tree_level_i$:	tree level of neighbor <i>i</i>		
$hops_i$:	hops to neighbor <i>i</i>		

Each node updates its neighbor list (Table 2) upon the reception of each Hello message. Not only the source of the Hello message is added into the neighbor list, but also the one-hop neighbors of the source³, with the *endAddr* and *tree_level* marked as *unknown* temporarily. The *unknown endAddr* and *tree_level* will be replaced with actual values when a Hello message is received from the corresponding neighbor. If no Hello message is received from some neighbors during the whole Hello message exchange procedure, a node can solicit for *endAddr* and *tree_level* information by broadcasting a message to its one-hop neighbors, including all the neighbors whose *endAddr* and *tree_level* are missing. Each one-hop neighbor received the message will reply if it can provide the *endAddr* and *tree_level* information of one or more neighbors included in the message. The field *hops* is computed according to the connectivity matrix described in subsection 3.3.2.

3.3.2 Connectivity Matrix

From the one-hop neighbor information included in each Hello message³, a node can construct a connectivity matrix for neighbors recorded in the neighbor list given in subsection 3.3.1. Table 3 illustrates one example.

The field *hops* of each node in the neighbor list can be calculated using the connectivity matrix. First the field *hops* of each node is set to *infinity*. Then, all nodes directly connected to *me* are one-hop neighbors (nb_2, nb_{n-1}, \dots in above example). Next, all nodes directly connected to one-hop neighbors (and having a *hops* of *infinity*) are two-hop neighbors (nb_1, nb_3, \dots in above example). This procedure continues until hop numbers of all neighbors are populated.

³The one-hop neighbor list included in an incoming Hello message with a TTL of 1 (i.e., this is the last hop of the Hello message) is not used for any purpose, since some of the nodes in the one-hop neighbor list may be (*maxHops* + 1) hops away from the receiver.

Table 1: Format of Hello Message

<i>begAddr</i>	<i>endAddr</i>	<i>tree_level</i>	<i>oh_nb₁</i> , <i>oh_nb₂</i> , ..., <i>oh_nb_k</i>
<i>begAddr</i> :	beginning address of the address block owned by the node (this is also the address assigned to the node itself)		
<i>endAddr</i> :	ending address of the address block owned by the node		
<i>tree_level</i> :	tree level of the node		
<i>oh_nb_i</i> :	address of one-hop neighbor <i>i</i> of the node		

Table 3: An Example of Connectivity Matrix

	<i>me</i>	<i>nb₁</i>	<i>nb₂</i>	<i>nb₃</i>	...	<i>nb_{n-2}</i>	<i>nb_{n-1}</i>	<i>nb_n</i>
<i>me</i>	-	-	+	-	...	-	+	-
<i>nb₁</i>		-	+	-	...	+	-	-
<i>nb₂</i>			-	+	...	-	-	-
<i>nb₃</i>				-	...	+	-	-
...				
<i>nb_{n-2}</i>						-	-	+
<i>nb_{n-1}</i>							-	-
<i>nb_n</i>								-

Note:

- (1) The plus or minus sign (“+” or “-”) at the cross cell of two nodes indicates they are or are not directly connected (i.e., they are or are not one-hop neighbors);
- (2) For bi-directional links, the matrix is symmetric, so only half of the matrix is needed as shown here.
- (3) Hop information can be calculated using the connectivity matrix. Here we have:
1-hop neighbors: *nb₂*, *nb_{n-1}*, ...
2-hop neighbors: *nb₁*, *nb₃*, ...
3-hop neighbors: *nb_{n-2}*, ...
4-hop neighbors: *nb_n*, ...

3.4 Data Forwarding

The pseudo code given in Table 4 elaborates on the procedure of selecting the next hop for data forwarding. When multiple neighbors are available for selection (see line 11 and line 32 of the pseudo code given in Table 4) and there are no other cost metrics indicating one neighbor is preferred over another, we can randomly select one neighbor for load balancing purpose. However, to mitigate “out of order” problems, it may be better to stick to one neighbor for a while once it is selected rather than randomly select one neighbor each time. If no next hop can be found, a ring search should be performed. Ring search can be done by exchanging Hello messages as in link state generation stage, but with an incremental TTL.

3.5 Sanity/Consistency Checking

To reduce communication overhead and interference, no periodic Hello messages are broadcast after the link state generation stage. During the data transmission stage,

Hello messages are only broadcast upon the detection of link failures, link recoveries, or new neighbors. If a node misses some Hello messages, its link state is likely to be inaccurate. Inaccurate link state can result in not only the selection of detoured routes but, more seriously, routing loops.

One way to promptly detect inaccurate link state without using periodic Hello messages is to include one-bit *up-down* flag and the so-called *virtual tree level* of the relaying node in each message being relayed. The *up-down* flag indicates whether the previous hop is forwarding the message up or down in terms of tree level. The *virtual tree level* is defined as follows:

$$vTL = \begin{cases} nbL - h2Nb & (if\ going\ down) \\ nbL + h2Nb & (if\ going\ up) \end{cases} \quad (1)$$

where,

nbL is the tree level of *nb_found* given in line 21 of the pseudo code in Table 4;

Table 4: Pseudo Code for Data Forwarding

```

1: func_nextHop(dst)
2:   nb_found = search nb list for the lowest (i.e., with the largest tree level)
           neighbor who is the ancestor of dst but is not my ancestor;
3:   if nb_found //going down
4:     next_hop = getOneHopNb(nb_found);
5:     return next_hop;
6:   else if dst is not my descendent //going up
7:     found = is there a neighbor who has a tree level less than mine?
8:     if found
9:       hops2root = the min (hops + tree_level) found among nb's that have
           a tree level less than mine;
10:      minHops = the minimum hops found among neighbors that have
           a (hops + tree_level) of hops2root;
11:      nb_found = select one of the neighbors that have a (hops + tree_level)
           of hops2root and a hops of minHops;
12:      next_hop = getOneHopNeighbor(nb_found);
13:      return next_hop;
14:     else //should go up, but can't
15:       return no_next_hop;
16:     end if
17:   else //should go down, but can't
18:     return no_next_hop;
19:   end if
20: end func

21: func_getOneHopNeighbor(nb_found)
22:   mark the hop_number of each neighbor as infinity;
23:   current_hops = hop number of the nb_found;
24:   while current_hops > 1
25:     for each neighbor nbi with a hop_number of current_hops
26:       for each neighbor nbj directly connected to nbi
27:         hop_number of nbj = current_hops - 1;
28:       end for
29:     end for
30:     current_hops = current_hops - 1;
31:   end while
32:   return one of the neighbors with hop_number of 1;
33: end func

```


To address the scalability problem, five sets of scenarios are defined, all in a grid form:

- 49 nodes (7×7 grid)
- 100 nodes (10×10 grid)
- 196 nodes (14×14 grid)
- 400 nodes (20×20 grid)
- 784 nodes (28×28 grid)

The distance between two horizontal or vertical neighbors is 10 meters. The PAN coordinator (i.e., the tree root) locates at the center of the network and starts a PAN at time 0.0 second. Every other node joins the PAN at a time randomly chosen between 0.0 and 5.0 second. The parameter *maxHops* takes the value 3. The radio propagation model adopted in all experiments is two-ray ground reflection⁷. And the transmission range is 12 meters. Constant bit rate (CBR) traffic is used and the packet rate is 1 packet per second as suggested in [7], with a packet size of 127 bytes at the PHY layer. Each 10 seconds a traffic flow is set up between two randomly selected nodes, and the duration of each traffic flow is determined in such a way that 5% of nodes in the network are transmitting data and another 5% are receiving data at any instant except the beginning and ending period of the simulation run. The total simulation duration is 2000 seconds, and the application traffic runs from 100 to 1900 second, leaving enough time for the experiment to shut down gracefully. Each experiment runs 10 times with different random seeds.

4.2 Hidden Terminal Issue

IEEE 802.15.4 suffers from hidden terminal problems as a result of lacking request-to-send (RTS) and clear-to-send (CTS) control messages [7]. Hidden terminal problems lead to the dropping of data packets and, more seriously, the tearing down of routes. A partial remedy is to apply a link failure threshold at an upper layer [3] and do not bring down a route before the failure threshold is reached.

In excess of the link failure threshold, we also use a scheme called receiver oriented TDMA (ROT) to improve the performance of IEEE 802.15.4. In ROT, when a node receives a Hello message, which contains the one-hop neighbor list of the source, from a one-hop neighbor, it adds the neighbor into its neighbor list and also calculates the time slot and slot cycle it should use to transmit packets to this neighbor. A simple way to calculate the time slot is to sort IDs/addresses of all the one-hop neighbors

⁷While a more realistic radio propagation model such as log-normal shadowing or Rayleigh fading can be used, using a deterministic radio propagation model like two-ray ground reflection provides a common non-time-varying condition for comparing different routing approaches, thus more accurately capturing the differences among them.

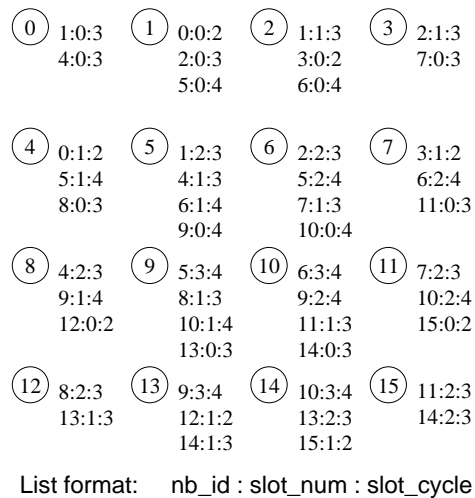


Figure 5: An Example of Receiver Oriented TDMA

of the neighbor where the Hello message comes in. Based on the order the node's ID/address appears in the sorted ID/address list, the node knows its time slot and also the slot cycle, which is just the number of one-hop neighbors included in the Hello message. For example, in Figure 5, node 5 has four one-hop neighbors (1, 4, 6, 9). The two-hop view of node 5 is (1 (0, 2, 5), 4(0, 5, 8), 6(2, 5, 7, 10), 9(5, 8, 10, 13)). Thus, node 5 has a time slot table (TST), (1:2:3/4:1:3/6:1:4/9:0:4), each entry of which is in the format of *neighbor_id:slot_number:slot_cycle*. The TST of node 5 tells that node 5 should use slot 2 (the third slot – slots are numbered from 0) (modulo slot cycle 3), 1 (modulo slot cycle 3), 1 (modulo slot cycle 4), and 0 (modulo slot cycle 4) to transmit packets to neighbor 1, 4, 6, and 9 respectively.

Each time a node receives a Hello message, it checks if the TST needs to be updated. Synchronization is needed in ROT, as in any other TDMA scheme. But as we have noticed, ROT is a fully distributed receiver oriented scheme, which implies no network-wide synchronization is needed. So the synchronization problem is simply reduced to that a node should know the clock of each its neighbor. Therefore, the synchronization problem can be easily solved by including the clock information in the Hello message. To compensate for clock inaccuracy and clock drift, some small guard time duration (GTD) can be added into each time slot and re-synchronization should be performed before the clock drift exceeds the GTD. Notice that the synchronization clock, slot number, and slot cycle used for one neighbor is independent to those for other neighbors.

In TDLS, the setup of ROT does not incur additional communication overhead except for the inclusion of clock

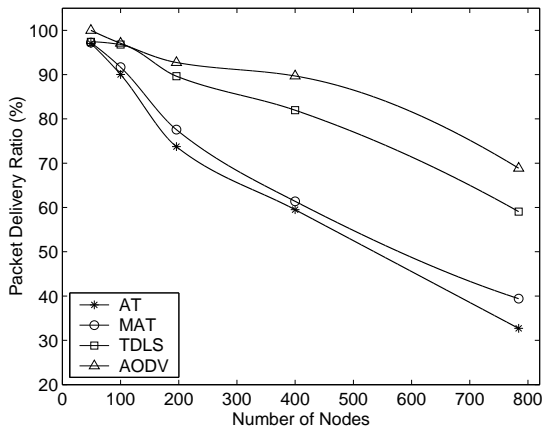


Figure 6: Packet Delivery Ratio

information in the Hello message, since it can be done together with the link state generation (see subsection 3.3). To avoid unnecessary delay when traffic load is light, ROT is only used for retransmissions. ROT is adopted in all our simulations. Our simulation results show that the performance improvement by utilizing ROT is significant, notwithstanding that this overly simple scheme can only prevent collisions between two or more packets having a common destination.

4.3 Numerical Results

In terms of packet delivery ratio (Figure 6)⁸, AODV outperforms all other three routings, namely, AT, MAT, and TDLS. When the number of nodes is larger than 400, AODV has a gain of about 8% in packet delivery ratio, when compared with TDLS. When the number of nodes is less than 200, both AODV and TDLS have a packet delivery ratio larger than 90%. For all scenarios, TDLS performs better than AT and MAT. And MAT in turn enjoys a slightly higher packet delivery ratio compared with AT. While the number of nodes roughly doubles each time, the packet delivery ratio drops smoothly, showing all schemes scale well in this regard.

The simulation results for end-to-end hop count are given in Figure 7. Surprisingly, AODV loses the battle to TDLS in this case. In fact, the performance of AODV is quite disappointing. Except for the 49 and 100 node scenarios, the end-to-end hop count of AODV is almost the same as that of AT or MAT. It is well known that tree routing such as cluster tree [23], AT, and MAT suffers from non-optimal routing paths. In contrast, AODV is claimed to be able to provide shortest routing paths in most cases.

⁸All figures in this subsection are plotted using piecewise cubic Hermite interpolation.

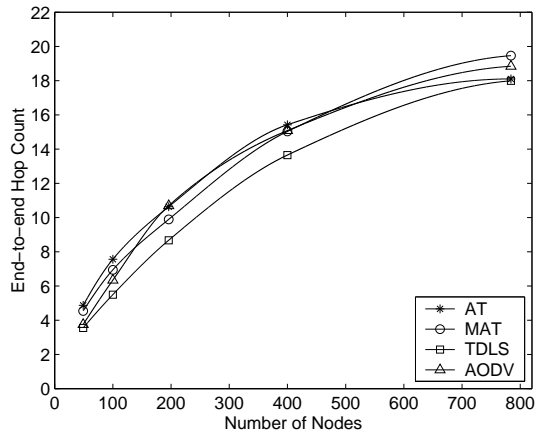


Figure 7: End-to-end Hop Count

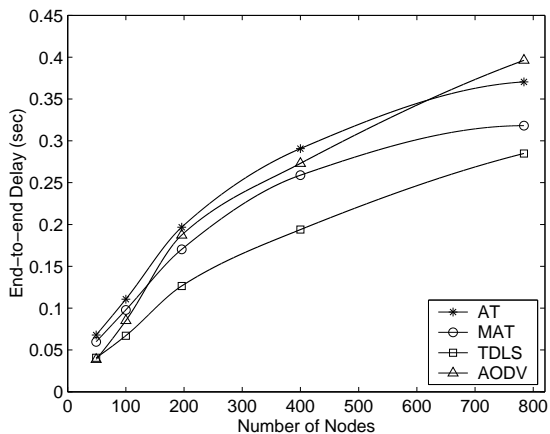


Figure 8: End-to-end Delay

Yet AODV falls short of achieving this goal in our simulations. By further delving into our simulation results, we find that flooding, which is used in AODV for route discovery and route repair, is to be blamed. Flooding is very detrimental in a wireless network, particularly one that lacks efficient means for coping with hidden terminal problems, as the case of LR-WPANs [7]. Although some reliable broadcast schemes have been proposed [3, 31], most broadcast schemes are unreliable. As a consequence, broadcast packets are often dropped due to collisions. To this end, it is of no surprise that AODV often misses the shortest path when the network traffic load is not light. As a matter of fact, sometimes AODV can not find a path at all. All our proposed schemes do not rely on flooding, though limited broadcast is sometimes used, mostly in a proactive way.

For both end-to-end delay and hop delay (Figure 8 and

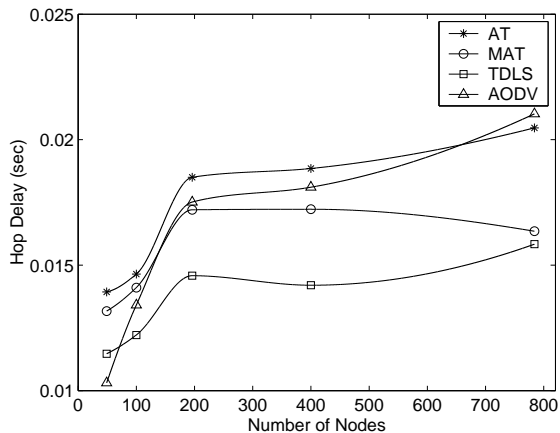


Figure 9: Hop Delay

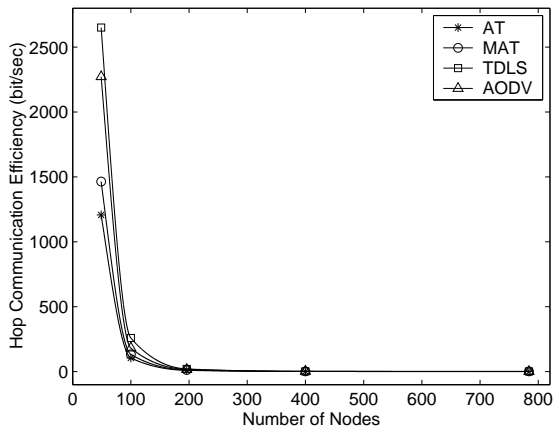


Figure 11: Hop Communication Efficiency

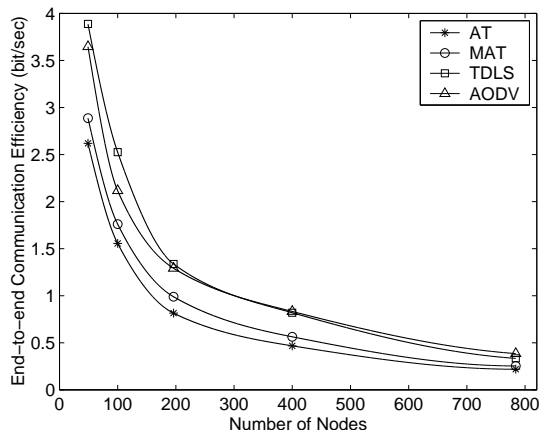


Figure 10: End-to-end Communication Efficiency

Figure 9), TDLS excels again. Followed is MAT, who performs better than AT and AODV. As one can see from Figure 8, the end-to-end delay of AODV begins to shoot up as the number of nodes reaches about 800. In terms of Hop delay, AODV peers AT, but gives in as the number of nodes exceeds about 700.

For end-to-end communication efficiency and hop communication efficiency (Figure 10 and Figure 11). The four routings rank in the order TDLS, AODV, MAT, and AT. For end-to-end communication efficiency (Figure 10), TDLS beats AODV when the number of nodes is less than 200, but has a similar performance as the number of nodes continues to climb. This is also true with the hop communication efficiency (Figure 11). It is noteworthy that the hop communication efficiency for all routings plumbs as the number of nodes increases from 49 to 100, or equivalently, from Figure 7, the end-to-end hop count

increases from about 4.3 to 6.3. The results indicate that, for the sake of communication efficiency, the end-to-end hop count is better to be limited to around 4.

5 Summary

An efficient scalable wireless mesh routing protocol, called topology-guided distributed link state (TDLS), is proposed in this paper. TDLS comprises two basic schemes, namely, the adaptive block addressing (ABA) scheme and the distributed link state (DLS) scheme. The ABA scheme is in charge of network auto-configuration and logic address assignment. It is also a self-contained routing protocol, which we call adaptive tree (AT) or, in its meshed form, meshed AT (MAT) routing. The DLS scheme utilizes the address block information provided by the ABA scheme as a guideline to extract the next hop for relaying a data packet. Compared with AT and MAT, it is able to render better performance in terms of hop count or other routing cost metrics used, robustness, and load balancing. Our simulation results also show that the TDLS scheme outperforms AODV in almost every respect under the scenarios used in simulation, notwithstanding its simplicity. One exception is that, when the number of nodes is more than 400, AODV has a gain of about 8% in packet delivery ratio compared with TDLS.

TDLS is far more memory-efficient than AODV, which makes it suitable for WMPANs and WMSNs that comprise devices with small RAM size. Although we evaluated AODV in networks with a node number up to 784 in section 4, in reality, AODV, as it is, would not have a chance in those networks, given the mere 10KB RAM owned by many 8-bit processors. Another point is that, as a proactive routing protocol, TDLS is suitable for time critical sensor applications, where again there is no room

for reactive routing protocols such as AODV.

References

- [1] IEEE P802.15.4/D18, Draft Standard: *Low Rate Wireless Personal Area Networks*, Feb. 2003.
- [2] ZigBee Alliance, <http://www.zigbee.org>.
- [3] ZigBee Network Specification, V1.0, Dec. 2004.
- [4] USC Information Sciences Institute, Marina del Rey, CA. *Network Simulator – NS2*. (<http://www.isi.edu/nsnam/ns>).
- [5] I. F. Akyildiz, X. Wang, and W. Wang, “Wireless mesh networks: a survey,” *Computer Networks Journal (Elsevier)*, Vol. 47, No. 4, pp. 445-487, Mar. 2005.
- [6] J. Zheng and M. J. Lee, “Will IEEE 802.15.4 make ubiquitous networking a reality?: A discussion on a potential low power, low bit rate standard,” *IEEE Communications Magazine*, Vol. 42, Issue 6, pp. 140-146, Jun. 2004.
- [7] J. Zheng and M. J. Lee, “A Comprehensive Performance Study of IEEE 802.15.4,” *Sensor Network Operations*, IEEE Press, 2005.
- [8] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on demand distance vector (AODV) routing,” IETF RFC 3561, Jul. 2003.
- [9] I. Chakeres and L. Klein-Berndt, “AODVjr, AODV simplified,” *ACM SIGMOBILE Mobile Computing and Communications Review*, pp. 100-101, Jul. 2002.
- [10] D. Johnson, D. A. Maltz, and J. Broch, “The dynamic source routing protocol for mobile ad hoc networks,” IETF Internet draft, draft-ietf-manet-dsr-09.txt, Apr. 2003.
- [11] T. Clausen and P. Jacquet, “Optimized link state routing protocol,” IETF RFC 3626, Oct. 2003.
- [12] R. G. Ogier, F. Templin, and M. Lewis, “Topology broadcast based on reverse-path forwarding (TBRPF),” IETF RFC 3684, Feb. 2004.
- [13] J. Moy, “Open Shortest Path First Routing Protocol,” IETF RFC 2328, Apr. 1998.
- [14] G. Malkin, “The routing information protocol,” IETF RFC 2453, Nov. 1998.
- [15] H. Frey, “Scalable geographical routing algorithms for wireless ad hoc networks,” *IEEE Network*, Vol. 18, Issue 4, Jul.-Aug. 2004.
- [16] B. Karp and H. T. Kung, “GPSR: Greedy perimeter stateless routing for wireless networks,” *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, Harvard University.
- [17] G. Pei, M. Gerla, and T.-W. Chen, “Fisheye state routing: A routing scheme for ad hoc wireless networks,” *Proc. ICC*, New Orleans, LA, Jun. 2000.
- [18] Z. J. Haas and M. R. Pearlman, “The zone routing protocol for ad hoc networks,” IETF: draft-ietf-manet-zone-zrp-02.txt, Jun. 1999.
- [19] G. Pei, M. Gerla, X. Hong, and C.-C. Chiang, “A wireless hierarchical routing protocol with group mobility,” *IEEE WCNC’99*, New Orleans, LA, Sep. 1999.
- [20] M. Joa-Ng and I-Tai Lu, “A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks,” *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 8, Aug. 1999.
- [21] M. Jiang, J. Li, and Y. C. Tay, “Cluster based routing protocol,” IETF draft: draft-ietf-manet-cbrp-spec-01.txt, Aug. 1999.
- [22] E. M. Belding-Royer, “Multi-level hierarchies for scalable ad hoc routing,” *ACM/Kluwer Wireless Networks*, 9 (5), 2003, pp. 461-478.
- [23] L. Hester, Y. Huang, A. Allen, O. Andric, P. Chen, “neuRFon Netform: A Self-Organizing Wireless Sensor Network,” *Proceedings of the 11th IEEE IC-CCN Conference*, Miami, Florida, Oct. 2002.
- [24] K. Carlberg and J. Crowcroft, “Building shared trees using a one-to-many joining mechanism,” *ACM SIGCOMM Computer Communication Review*, Vol. 27, Issue 1, pp. 5-11, 1997.
- [25] J.-J. Pansiot and D. Grad, “On routes and multicast trees in the Internet,” *ACM SIGCOMM Computer Communication Review*, Vol. 28, Issue 1, pp. 41-50, 1998.
- [26] R. G. Ogier, “Efficient routing protocols for packet radio networks based on tree sharing,” *Proc. Sixth IEEE Intl. Workshop on Mobile Multimedia Communications (MOMUC’99)*, Nov. 1999.
- [27] C.-C. Chiang, M. Gerla, and L. Zhang, “Adaptive shared tree multicast in mobile wireless networks,”

IEEE Communications Magazine, Vol. 3, pp. 1817-1822, Aug. 1998.

- [28] N.-F. Tzeng and P. Alla, "Guided shared trees for efficient multicast in large networks," *Proc. IEEE Int'l Conference on Communications (ICC 2003)*, May 2003.
- [29] Sajama and Z. J. Haas, "Independent-tree ad hoc multicast routing (ITAMAR)," *Mobile Networks and Applications*, Vol. 8, Issue 5, pp. 551-566, 2003.
- [30] Amtel Corporation. <http://www.atmel.com>
- [31] C. Zhu, M. Lee, and T. Saadawi, "A border-aware broadcast scheme for wireless ad hoc network," *IEEE Consumer Communications and Networking Conference*, 2004.