IEEE P802.15
Wireless Personal Area Networks

Project	IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)		
Title	**IEEE 802.15.5 WPAN Mesh Networks**		
Date Submitted	[May 13, 2005]		
Source	[**Jianliang Zheng, Yong Liu, Chunhui Zhu, Marcus Wong, Myung Lee**] [Samsung Lab @ CUNY] [Steinman Hall #677, 140th St & Convent Ave, NY, NY 10031, USA]	Voice: Fax: E-mail:	[+1-212-650-7260] [+1-212-650-8249] [lee@ccny.cuny.edu]
Re:	[IEEE P802.15-5/0071]		
Abstract	This proposal discusses Samsung's proposal for IEEE 802.15.5 WPAN Mesh, based on Meshed-Tree approach, including Meshed Tree routing, multicasting, and Key pre-distribution		
Purpose	This proposal is provided to be adopted as a recommended practice for IEEE WPAN Mesh		
Notice	This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.		
Release	The contributor acknowledges and accepts that this contribution becomes the property of IEEE and may be made publicly available by P802.15.		

IEEE 802.15.5 WPAN Mesh Networks

This document describes a proposal for IEEE 802.15.5 WPAN Mesh proposal submitted by Samsung Lab at City University of New York.

The contents included in this document are:

- General Meshed Tree
- Topology Server Based Meshed Tree, especially for beacon enabled networks
- Tree based multicast protocol
- Key pre-distribution algorithm

Meshed Tree

1. Introduction

Tree is a useful structure for routing purpose. The adaptive robust tree (ART) and its meshed form, meshed ART (MART), proposed here enable a node to find out the next hop by checking the destination address in a packet. This precludes the need of route discovery, and thus helps reduce the initial latency, control overhead, memory consumption and energy consumption. Compared with other tree routing, ART has the following unique features: (1) adaptive logic address assignment; (2) efficient handling of single point of failure (SPOF); (3) no logic address change during tree repair; (4) increased robustness and route optimization achieved by using meshed ART (MART).

MART can provide better routes than ART; nonetheless routes in MART are still non-optimal in most cases. To optimize routes, we further combine ART/MART with another on-demand non-tree routing. While packets can always be transmitted via the available ART/MART route, the source can optionally trigger a route discovery procedure to find an optimal non-tree route to the destination. Tree routes along a single tree branch are generally optimal, if the tree has been optimized with respect to the routing cost(s) under consideration (e.g., hop count, link quality, and power)[1]. Based on the above, optimal non-tree routes are expected to be orthogonal with high probability to tree routes in the sense that they mainly connect different tree branches. As a result, tree routes and non-tree routes interconnect all nodes and form a mesh.

[1] Tree optimization can hardly be done when link quality is used as a routing cost. Generally there is little data traffic during the tree formation period, and therefore the link quality measured during this period does not reflect the real link status. In fact, it is not the link quality that determines the tree structure; rather it is the tree structure that significantly affects the link quality. However, considering that, along a tree branch, a node closer to the root normally has a worse tree link and that tree links of sibling nodes tend to have similar link quality due to the interference between them, it still holds that tree routes along a single tree branch are generally optimal.

2. Adaptive Robust Tree (ART)

2.1 ART table

Table 1: Adaptive Robust Tree Table (ARTT)

num_branch				
$type_1$	beg_addr_1	end_addr_1	$priority_1$	$next\_hop_1$
$type_2$	beg_addr_2	end_addr_2	$priority_2$	$next\_hop_2$
……				

num_branch: number of branches the node has
$type_i$: type of branch *i* (*desIn*, *desOut*, *srcIn*, *srcOut*)
beg_addr_i: beginning address of branch *i*
end_addr_i: ending address of branch *i*
$priority_i$: priority of branch *i* (*normal*, *high*)
$next\_hop_i$: next hop via which the whole branch *i* is routed

In adaptive robust tree (ART), each node keeps an ART table (ARTT) to track its branches. The structure of the table is given in Table 1. Each branch is assigned a block of consecutive addresses. But there is no need to assign consecutive blocks to those branches. This feature allows adding new branches later as well as repairing the tree efficiently. Field $type_i$ determines under what condition a branch is selected for relaying a packet. For example, if $type_i$ = *desIn*, then the branch will be selected for relaying a packet if the destination address of the packet falls in the address block [beg_addr_i, end_addr_i]. In case that the destination address (or source address) of a packet falls in (or out of) multiple address blocks, priority is applied in the order *desIn*→*desOut*→*srcIn*→*srcOut*; and field $priority_i$ further distinguishes branches of the same type. The parent of a node is treated as an implicit branch (i.e., not recorded in the ARTT of the node), which has the lowest priority. The function of ARTT will be discussed in detail in the following subsection.

2.2 Three phases

We define three phases for ART: initialization (or configuration) phase, normal phase, and recovery phase.

1) During initialization phase, nodes join the network and a tree is formed.
2) After initialization, the network enters normal phase, in which normal communications start. During normal phase, new nodes are still allowed to join the network, but the number of new nodes should be small compared with the number of nodes already in the network. If there is a big change of either the number of nodes or the network topology, the initialization should be done again. That is, the network needs to be re-configured.
3) If the tree is broken, then the recovery phase is triggered. Note that recovery phase is different from the other two phases in that only the affected part of the tree needs to enter the recovery phase (other unaffected part is still in normal phase). And only

failures (either node failures or link failures) happened after initialization will trigger recovery phase. Failures during initialization do not trigger recovery phase, and they should be handled by initialization phase itself.

2.2.1 Initialization phase

A tree is formed during initialization phase. ART tree formation is functionally divided into two stages: association and address assigning.

During association stage, beginning from the root (normally designated manually, for example, by pressing a button), nodes gradually join the network and a tree is formed. But this tree is not an ART tree 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) it will accept according to its capability and other factors. However, it is possible that a node cannot find any other node to associate if the number of children 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 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 node receives multiple association responses, it should choose the node with the highest AD for association, unless there are other high priority factors indicate not to do so. Using AD increases the chance with which a node successfully joins the network without severely overloading an individual node.

After a branch reaches its bottom, that is, no more nodes wait for joining the network (a suitable 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 square brackets indicate the numbers of nodes within branches below a certain node.

When numbers of nodes are reported from the 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*) receives the information from all the branches, it will begin to assign addresses.

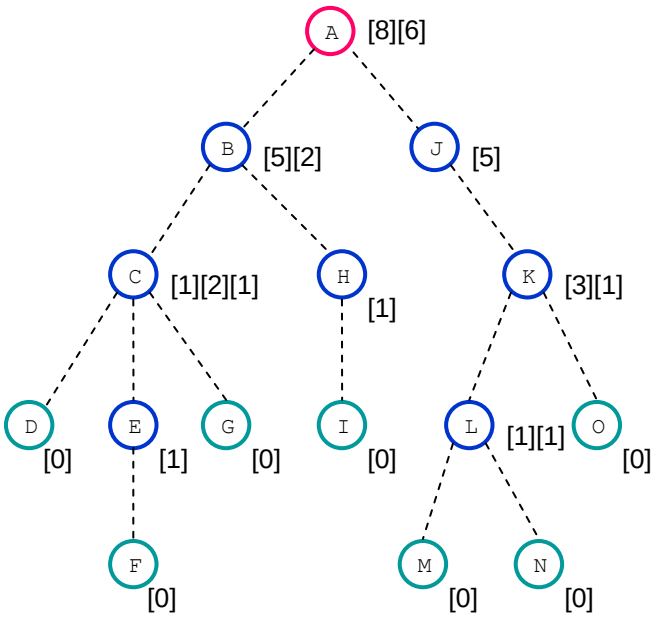

Figure 1: Calculation of number of nodes along each branch

During address assigning stage, a top-down procedure is used. First of all, the root will check if the total number of nodes in the network is less than the total number of addresses available. If not, address assignment fails. The possible solution in case of address overflow is to separate the network into smaller ones or increase the address space. Here we will not handle the address overflow, and will assume no address overflow happens. Next, the root will assign a block of consecutive addresses to each branch below it, taking into account the actual number of nodes and the wished number of addresses. Note that the actual number of addresses assigned could be less than the wished one (but no less than the actual number of nodes) or more than the wished one, depending on the availability of addresses. This procedure will continue until the bottom of the tree. After address assigning, an ART tree is formed and each node has an ARTT for tracking branches below it. For example, node *C* can have an ARTT somewhat like:

[3] //3 branches
[*desIn*][6][8][*normal*][6] //first branch owns address block [6, 7, 8],
//and packets should be routed through
//itself (*next_hop*$_i$=6 points to branch 1 itself).
[*desIn*][9][13] [*normal*] [9] //second branch owns address block
//[9, 10, 11, 12, 13], and packets should be routed
//through itself
[*desIn*][14][14] [*normal*] [14] //third branch owns address block [14]
//(no additional addresses reserved), and packets
//should be routed through itself

2.2.2 Normal phase

After an ART is formed, the network enters normal phase. Using ART, a packet can be easily routed. For example, when a packet is received by node *C*, node *C* will check if the destination belongs to one of its branches. If the packet belongs to branch *i*, the packet will be relayed to the node with address $next\_hop_i$. Note that $next\_hop_i$ is not necessarily equal to beg_addr_i. The use of $next\_hop_i$ is for handling tree repair. If the destination does not belong to any branch, it will be routed to the parent of node *C*.

Although no significant change of the number of nodes or the network topology is expected during normal phase, it is still allowed to add more nodes (therefore, branches) at any level of the tree, only if additional addresses (reserved during initialization phase) are available. If there is a big change of the number of nodes or network topology, which cannot be handled during normal phase, the network can go through the initialization phase again.

2.2.3 Recovery phase

During normal phase, link failures will trigger recovery phase. Route repair (for both tree link and non-tree link) will be described in subsection 3.4.

2.3 Meshed ART (MART)

With a MART, we can do two things. First, it is possible to route a packet through a shorter path. Second, it is possible to eliminate some SPOFs.

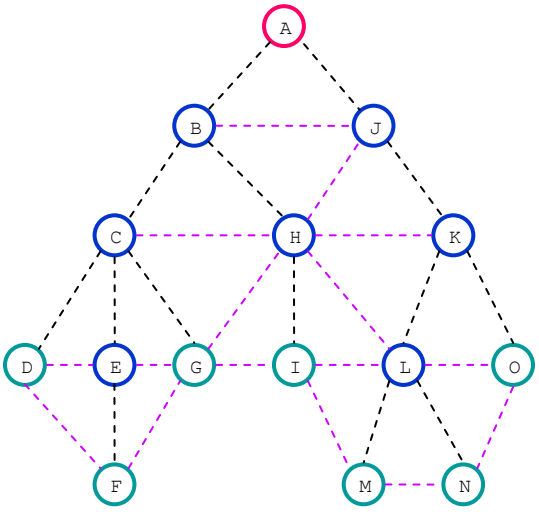

Figure 4: Meshed ART

After an ART tree is formed, we can further form a meshed ART (Figure 4). The original ART tree is connected through black lines. Additional magenta lines 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 through a magenta line will treat each other

as a child and add an ARTT entry for each other. For example, node *K* will treat node *H* as a child, and vise versa.

By forming an MART, it is possible to route a packet through a shorter path (compared with pure ART). For example, a packet from node *M* to node *I* can be transmitted directly from node *M* to node *I*, since from node M's point of view, node *I* is its child. On the other hand, the normal path 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 used. For example, from node *E* to node *H*, the path will be *E-C-H* instead of *E-C-B-H*. Note that the path from node *K* to node *G* is not *K-H-G*, but *K-J-B-C-G*. The reason is that, although node *H* is treated as a descendent of node *K* under MART, node *G* is not. For simplicity, node *H* only gives node *K* the ART address block of itself, not including other address blocks added either due to tree repair or due to meshing behavior. So node *K* will not forward a packet destined for *G* directly to node *H*.

Another advantage is that some SPOFs 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. But packets from node *K* to other nodes such as node *B* cannot be routed before tree repair is done.

3. Mesh Wireless Personal Area Networks (MWPAN)

3.1 Routing Table

Table 2: Non-Tree Table (NTT)

beg_addr_1	end_addr_1	$next\_hop_1$	$hops_1$	$cost_1$	$tstamps_1$
beg_addr_2	end_addr_2	$next\_hop_2$	$hops_2$	$cost_2$	$tstamps_2$

num_entry:	number of NTT entries
beg_addr_i:	beginning address of entry *i* (it is also the destination address)
end_addr_i:	ending address of entry *i*
$next\_hop_i$:	next hop via which entry *i* can be routed
$hops_i$:	hops to the beginning address of entry *i*
$cost_i$:	cost to the beginning address of entry *i* (no need if hop count is the only cost)
$tstamp_i$:	time when entry *i* is created or refreshed

Besides ARTT, another type of table called non-tree table (NTT) is defined in MWPAN (Table 2). Each NTT entry provides not only an optimal route to address beg_addr_i, but also an auxiliary route to the whole address block [beg_addr_i+1, end_addr_i], if $end\_addr_i > beg\_addr_i$. An auxiliary route may or may not be optimal, but it is in general better than the corresponding non-optimal MART route. In terms of cost, roughly we have:

$$ART\_route \geq MART\_route \geq NT\_auxiliary\_route \geq NT\_optimal\_route$$

Whenever a route is optimal, equal sign applies from there on.

3.2 Data Forwarding

In MWPAN, a local repair scheme is used for repairing broken links (see subsection 3.4). For simplicity and practically, a locally repaired tree route is still deemed optimal if the original one is optimal. This implies that any tree route recorded in an ARTT is deemed optimal. Therefore, a data packet can be forwarded using a route selected according to the flow depicted in Figure 5.

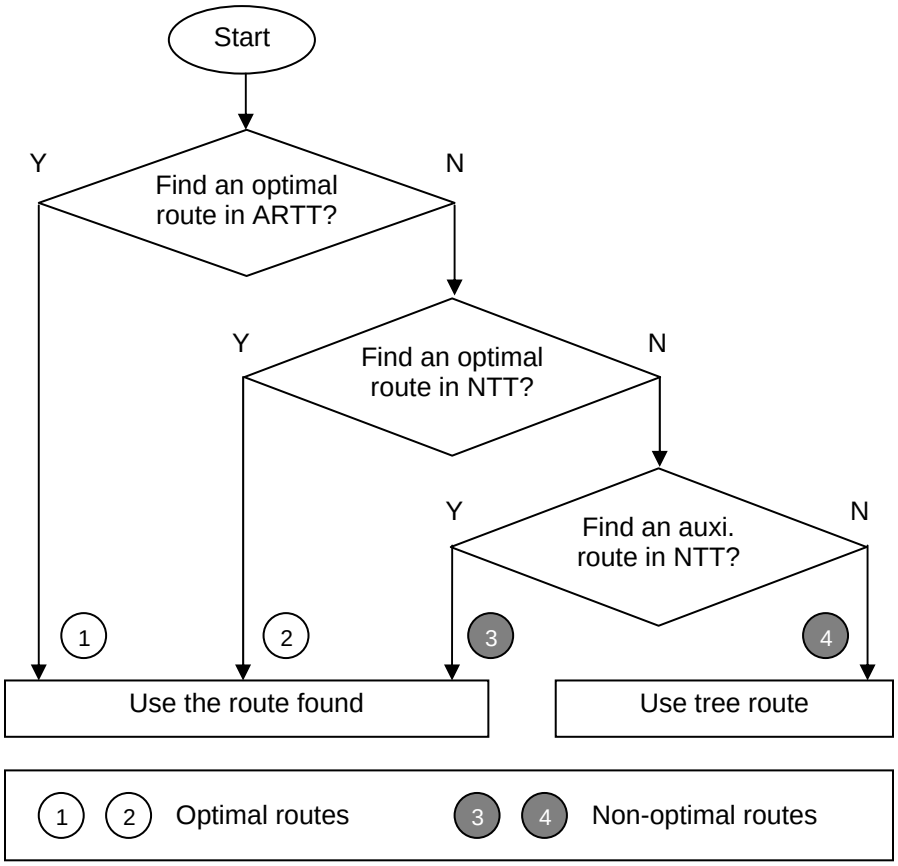

Figure 5: Data Forwarding

3.3 Route Discovery

Table 3: Route Request (RREQ) and Route Reply (RREP) Packet Formats

RREQ				
route_type	*dst_beg_addr*	*src_beg_addr*	*src_end_addr*	*max_link_cost*
hops_traveled	*cost_accumed*	*TTL*		

RREP				
route_type	*dst_beg_addr*	*dst_end_addr*	*src_beg_addr*	*src_end_addr*
hops_traveled	*hops_total*	*cost_accumed*	*cost_total*	*TTL*

route_type: route type (*optimal_NT*, *optimal_ART*, *non-optimal*, *unknown*)
dst_beg_addr: beginning address of destination tree branch
dst_end_addr: ending address of destination tree branch
src_beg_addr: beginning address of source tree branch
src_end_addr: ending address of source tree branch
max_link_cost: maximum link cost along the propagation path of the RREQ (used to filter out routes whose *max_link_cost* exceed a certain threshold, i.e., the RREQ will no longer be relayed if *max_link_cost* recorded in it exceeds a certain threshold)
hops_traveled: hops the RREQ/RREP has traveled
hops_total: total hops between the source and the destination (no larger than the *hops_traveled* in the RREQ unicast from the source)
cost_accumed: cost the RREQ/RREP has accumulated
cost_total: total cost between the source and the destination (it is no larger than the *cost_accumed* in the RREQ unicast from the source; if asymmetric links are assumed, *cost_total* is not needed and optimal routes for different directions should be discovered separately)
TTL: Time to live

When no optimal route is available to a destination, the source may trigger a route discovery procedure to find one, or just use the best available non-optimal route to relay packets when traffic duration is short or the available route is near-optimal. Route discovery proceeds as follows[2]:

1. The source initiates the route discovery procedure by unicasting a route request (RREQ) packet, whose format is given in Table 3, to the destination along the best available route. After receiving the unicast RREQ, the destination will first check if it has an optimal route to the source[3] according to the flow shown in Figure 5. If an optimal route is found, the destination will unicast a route reply (RREP) packet, whose format is given in Table

[2] Here all wireless links are assumed to be bi-directional.
[3] Note that the destination may have an optimal route to the source, even if the source does not have one to the destination. Following are some examples.

- The source is a descendent of the destination, but multiple tree levels away from the destination.
- The source is an end point of an optimal NTT route, but the destination is not.

3, to the source along this route. If *route_type = optimal_NT* (that is, an optimal non-tree route to the source is available), as the RREP travels from the destination to the source, each node along the path adds or refreshes two NTT entries, one for the source and one for the destination, unless an optimal tree route already exists[4].

2. If no optimal route to the source is found, the destination will then try to find an optimal route by broadcasting an RREQ with a time to live (TTL) equal to *hops_traveled*.
3. If neither a unicast RREP nor a broadcast RREQ is received from the destination within a certain timeout period after unicasting an RREQ, the source will flood an RREQ to find an optimal route to the destination.

A broadcast RREQ will be unicast to the destination when it reaches a tree branch to which the destination recorded in the RREQ belongs. But it may be broadcast again when tree link breaks.

3.4 Route Repair

Whenever a link is broken, the node having detected the problem will perform a local repair by broadcasting an RREQ with a TTL equal to *rpr_min_TTL*. If no RREP is received within a certain timeout period, the node increases the TTL by *rpr_inc_TTL* and tries again. This procedure continues until at least one RREP is received or the TTL reaches *rpr_max_TTL*. If multiple local routes are found, the node will select the best one based on the costs recorded in the corresponding RREPs. If the local repair is for a non-tree link, the node will unicast a gratuitous RREP along this local route so that all related nodes can also build a non-tree routing entry to the original source. If the local repair is for a tree link, a bi-directional ARTT routing path will be built between the two nodes originally connected via tree link (they could be multiple tree levels away). If the local repair fails, a route error (RERR) packet will be unicast to the source and, upon receiving this RERR, the source will try to rediscover an route to the destination. The suggested default values for *rpr_min_TTL*, *rpr_inc_TTL*, and *rpr_max_TTL* are 3, 1, and 2×*rpr_min_TTL*.

Figure 6 shows a tree link repair example. Node *K*, which is along the routing path from node *C* to node *M*, fails. Node *J* successfully finds another local route *J-H-L* by broadcasting an RREQ with a TTL equal to 3 (not all broadcast propagation is shown in the figure). To relay packets from node *J* to node *L*, following modifications are made (assume the address block of node *L* is [$addr_0$, $addr_m$]):

Node *J* adds a branch: [*desIn*][$addr_0$][$addr_m$][*high*][*H's address*]

Node H adds a branch: [*desIn*][$addr_0$][$addr_m$][*normal*][*L's address*]

To relay packets from node *L* to node *J*, following modifications are made:

Node *L* changes its parent from node *K* to node *H*.

Node *H* adds a branch: [*srcIn*][$addr_0$][$addr_m$][*normal*][*J's address*]

Note that not all data packets that leave branch *L* will be routed through the path *L-H-J*. For example, a data packet from node *M* to node *E* will actually follow the path *M-L-H-C-E*.

[4] An optimal tree route may be part of an optimal NTT route and, in this case, no NTT entries need to be created along this optimal tree route.

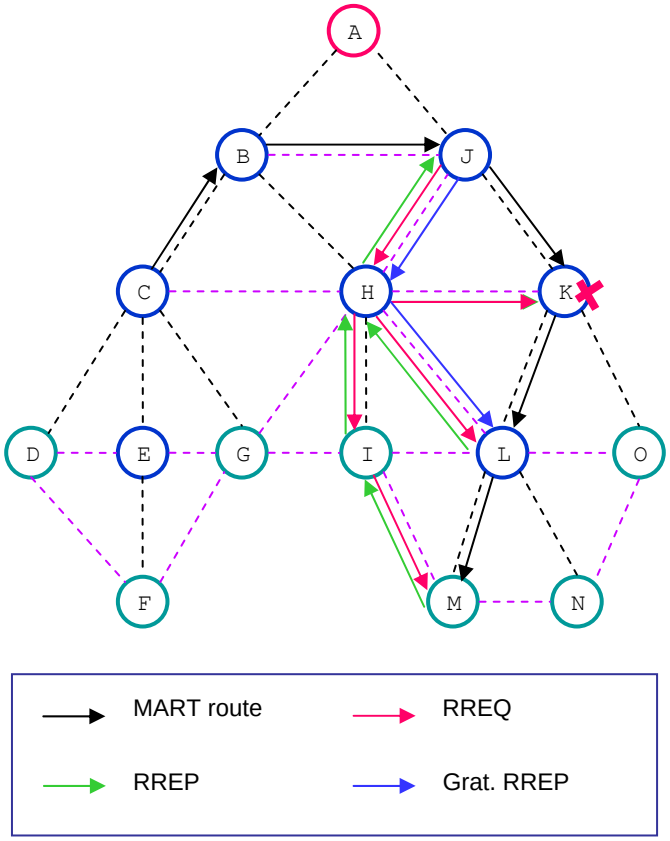

Figure 6: Route Repair

The above example only shows the repair for branch *L*. But actually one local repair will try to repair all detached branches.

Mesh Networking Designs for Low Duty Cycle Networks

Introduction

This document describes a mesh network layer built on top of IEEE 802.15.4 medium access control (MAC) and physical (PHY) layers. The functions of this mesh network layer include:

- Establishing a self-routing tree to cover the whole network
- Scheduling beacon transmissions at a topology server to avoid beacon collisions
- Calculating shortcuts between active source-destination pairs at a topology server
- Recovering from link/node failures by recalculating new routes or reforming the tree

Basic Mechanisms

Tree formation

The node initiating the network becomes the PAN coordinator. In the network formation stage, all coordinators shall enable their receivers to catch beacon requests from new nodes. No regular beaconing is allowed before the beacon scheduling is done. New nodes perform active scan to collect beacons from their neighbors and record the qualities of links to these neighbors. Every new node selects a neighbor, which has the best path quality to the PAN coordinator, as its parent, and associates with its parent. Eventually, all nodes are connected as a tree rooted at the PAN coordinator.

Tree addressing

Once the tree is formed, the PAN coordinator broadcasts a descendant statistics message along the tree to all leaf nodes. Each leaf node returns a descendant counter to its parent with the initial counter value set to one. After a coordinator collects the descendant counters from all its children, it adds them together and passes the sum to its own parent. This process continues until the PAN coordinator receives the descendant counters from all its children.

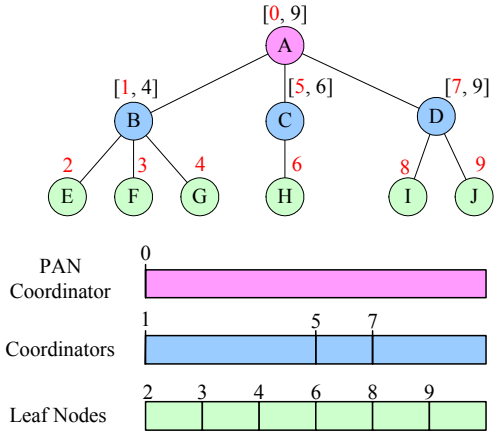

Figure 1. Tree addressing

The PAN coordinator divides the available address[5] block among its children based on the ratios of their descendant numbers. Each coordinator, once receiving the address block assigned by its parent, further divides the address block among its own children. This process continues until all leaf nodes receive their addresses. Figure 1 shows an example of the tree addressing.

Tree routing

When a coordinator receives a packet not destined for itself, it first checks whether the destination's address falls into its address block. If not, the packet is forwarded to its parent. Otherwise, it compares the destination's address with its children's addresses. The packet is forwarded to child *i* if the destination's address is between the addresses of child *i* and child *i*+1.

In the example of Figure 2, node H sends a packet to node F. As H does not have any child, it forwards the packet to its parent C. C finds the destination F has an address out of its address block. So it forwards the packet to its parent A. As F's address falls into A's address block, and A further finds that F's address is between the addresses of child B and child C, so A forwards the packet to B, and B forwards it to F.

[5] The address here is MAC short address

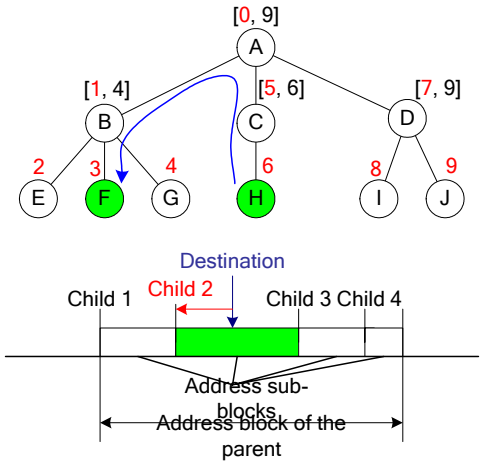

Figure 2. Tree routing

Topology server setup

Either the PAN coordinator or a resource sufficient node can serve as the topology server. All other nodes can reach the topology server by using tree routing. Every coordinator shall report its superframe parameters and link states to the topology server after the tree is formed. In order to make the topology server maintain up-to-date topology information, each coordinator may periodically scan its neighbors' beacons and report significant link changes to the server. There can be two or more topology servers acting as backup of each other.

Beacon scheduling

When receiving the neighboring information of a coordinator, the topology server assigns a contention-free beacon time-slot to the coordinator. Every coordinator gets a beacon time slot that is not overlapped with the active periods of its two-hop neighbors. This two-hop beacon scheduling ensures that each node can correctly capture all its neighbors' beacons and locate their active periods. Once a coordinator receives the beacon time assignment, it can emit regular beacons and operate in beacon-enabled mode.

Reactive shortcut formation

When an active source wants to find a shortcut to its desired destination, it sends a shortcut request (SCRQ) message to the topology server through the tree route. The topology server calculates the optimal shortcut between the source and the destination by using the Dijkstra's algorithm. After deriving the shortcut, the topology server sends a shortcut notification (SCNF) message to the destination with a list of relay nodes on the shortcut. The destination then generates a shortcut reply (SCRP) message and sends the message to the source along the newly derived path. The SCRP, in its propagation from the destination to the source, establishes both forward and backward routing entries in the relay nodes on the

shortcut. Figure 3 shows an example of the reactive shortcut formation. In the example, the PAN coordinator acts as the topology server.

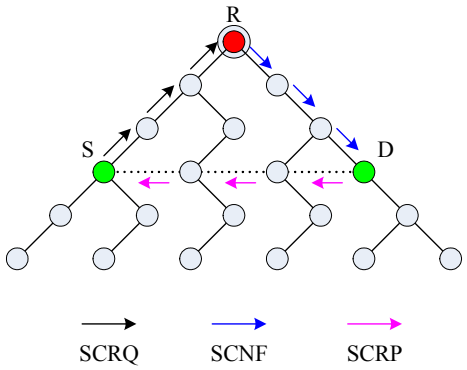

Figure 3. Reactive shortcut formation

To facilitate packet transmissions through the shortcut, relay nodes along the shortcut shall locate and record the active periods of their previous-hop and next-hop neighbors.

Two-address strategy

The tree-addressing scheme introduced in section 2.2 tightly relates the MAC short addresses with the tree structure. Once the tree structure changes, the affected nodes may change their MAC short addresses accordingly to maintain tree routing. To prevent the address changes from affecting existing shortcuts, the shortcut routing may use NWK addresses, instead of MAC short addresses. The NWK addresses, once assigned, will never be changed.

Each packet shall carry the destination's NWK address. When the packet is delivered through a tree route, it shall also carry the destination's MAC short address.

Route repair

If the topology server is not the PAN coordinator, it keeps maintaining the route between it and the PAN coordinator. So that the PAN coordinator can always reach the server.

When a node detects the failure of a non-tree link or a tree link to its child, it updates the link change to the topology server. If it cannot reach the topology server directly, it may ask the PAN coordinator to forward the link change to the server. The topology server shall recalculate a new route from this node to the destination.

When a node cannot reach its parent, it initiates tree reconstruction by dismissing all its descendants. All affected nodes rejoin the tree and obtain new MAC short addresses from their new parents (NWK addresses are not changed). The node detecting the link failure updates the

topology server about the link change after it rejoins the tree, and the topology server recalculates new routes from this node to affected destinations.

If there is an address-mapping server, the nodes with new MAC short addresses shall also update the address server.

The sources with the old addresses of desired destinations either contact the address server or broadcast an address search message along the tree to obtain the new addresses of the destinations.

The existing shortcuts are not affected since they depend on NWK addresses.

Enhancements

Solution to potential beacon collisions

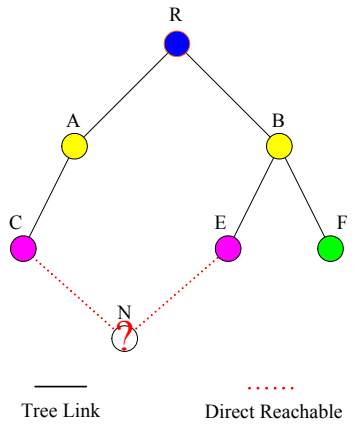

Figure 4. Beacon collision problem

As shown in Figure 4, when node N does not appear in the network, node C and E are not two-hop neighbors. The topology server may assign the same beacon time slot to them. When node N presents and tries to join the tree, it cannot receive beacons from either C or E. One way to alleviate this problem is to make new nodes scan not only beacons, but also other packets. According to the scanning results, the new nodes may estimate active periods of their neighbors, and request the neighbors to send ad hoc beacons. Another way is to make the topology server define an irregular beacon period that is not overlapped with the active period of any coordinator, so that each coordinator can send irregular beacons during this period in randomly selected superframes.

Solution to single point of failure

In order to avoid the single point of failure, an extra server can be set up as the backup of the main server. The main server shall synchronize its topology knowledge with the backup server, and the neighbors of the main server shall record the address of the backup server. Once the main server fails, all messages destined for the main server are forwarded to the backup server.

When the PAN coordinator acts as the topology server, the neighbors of the PAN coordinator may establish alternative table-driven routes to reach each other (bypassing the PAN coordinator). Once the PAN coordinator fails, its neighbors can still maintain the tree routing.

For example, in Figure 5, the PAN coordinator R has three children A, B, and C. Alternative routes are established among these children. When R fails, node A can forward packets to C's descendants through an alternative route A-B-C.

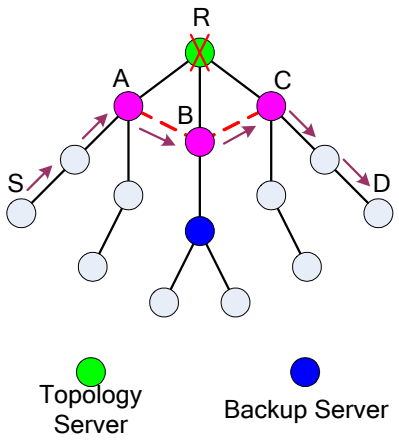

Figure 5. Solution to single point of failure

Solution to large network case

3.3.1 Partial topology server

If a topology server cannot accommodate the whole network topology, it may maintain only the topology around the root area, i.e. the top portion of the tree. This area is called server coverage area (SCA). The topology server can help nodes within the SCA establish optimal shortcuts to reach each other. Nodes outside the SCA have to use tree routes. If a tree route passes through the SCA, the topology server can optimize the part of the tree route lying within the SCA.

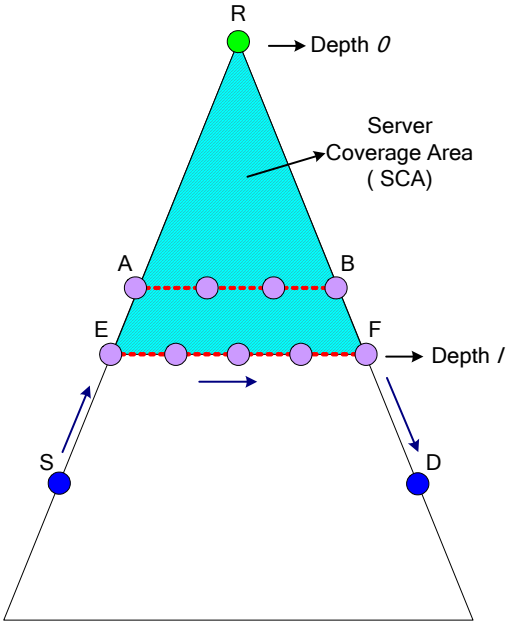

Figure 6. Partial topology server

Figure 8 shows an example of the partial topology server. Because of the limited capacity, the topology server R can only cover the nodes from depth *0* to depth *l*. Node R can help nodes A, B, E, and F establish the optimal routes among themselves, however, it cannot derive the optimal route between node S and node D. When node S delivers a packet to node D, it has to use the tree route first. Once the packet hits a node within the SCA, such as node E in this example, node E can serve as the agent of the original source S and seek helps from R to derive a shortcut to reach node D. Node R finds that, within the SCA, node F is the nearest ascendant of node D, so it calculates an optimal route between E and F. Node R then sends a SCNF message to node D with the shortcut information, and node D sends a SCRP back to node E to establish routing entries along the shortcut.

3.3.2 Multiple topology servers

If multiple nodes can act as topology servers, the network can be partitioned into several subtrees. Each subtree is assigned a local topology server. Communications within a subtree can be handled by the local topology server.

A central topology server is also named to manage the communications among different subtrees. The central topology server can help the roots of different subtrees establish shortcuts to connect each other.

Figure 7 shows an example of multiple topology servers. Node A, B, C, and D are the local topology servers of four subtrees. Node R is the central server covering the top portions of the four subtrees. Route A-C-F-D is an inter-tree shortcut derived by R.

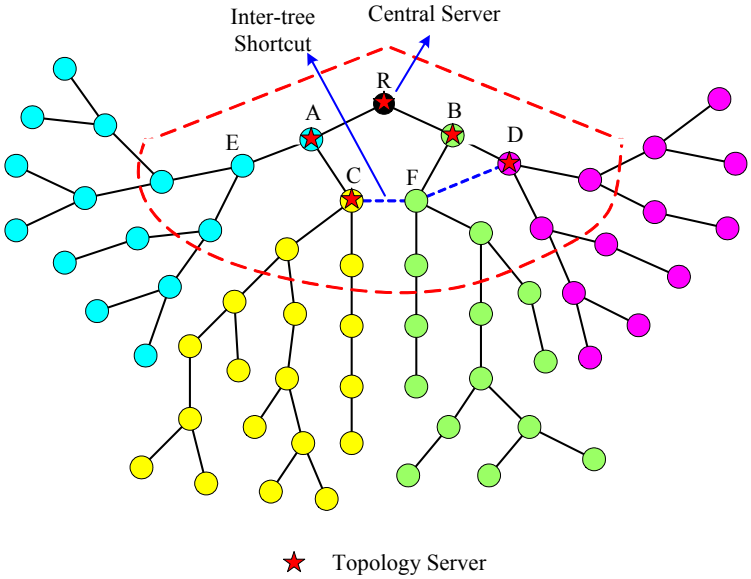

Figure 7. Multiple topology servers

A Multicast Routing Protocol Based on Adaptive Robust Tree (ART) for IEEE 802.15.5 WPAN Mesh Networks

1. Introduction

This proposal proposes a multicast routing protocol based on the Tree-Based Mesh Routing (TBOnTR) protocol we proposed in another related file. The TBOnTR protocol builds an Adaptive Robust Tree (ART) which covers all the nodes in a WPAN mesh network. The ART is a share tree rooted at the Tree Coordinator. When the tree is built, the neighbor information as well as its relationship to a node is recorded in every node's neighbor list.

Our goal of multicast routing hence can be defined as finding a minimum sub-tree of the Adaptive Robust Tree which covers all multicast members within each multicast group. The joining and leaving the multicast group is dynamic. A mechanism is provided to ensure the multicast tree is minimal at any time during the multicast session;

The algorithm features fast packet delivery, the least possible control traffic and very little memory requirement. When a multicast packet is to be sent from a group member, it will be propagated immediately to other members follow the multicast tree without being sent to the Tree Coordinator first. The packet transmission delay is then minimized.

Our general assumption for this proposal is the following.

- Both MAC address and logical short address can be used for routing purpose;
- There is a thin Mesh layer running on top of MAC layer for routing purpose;
- Necessary fields are available from the Mesh layer packet header.

2. The ART-based Multicast Routing

2.1 Definitions of the Protocol Entities

- **Group Member (GM)** – a node participating a multicast group
- **On-Tree Router (OnTR)** – nodes on the multicast tree but not GMs
- **Group Coordinator (GC)** – the top level GM or OnTR of a specific multicast group (sub-tree root). It sets the upper bound of the multicast tree.
- **Tree Coordinator (TC)** – the root of the ART. It keeps information of all multicast groups in the network so that it always knows from which child(ren) it can reach the multicast tree for a specific group.
- **Off-Tree Router (OffTR)** – nodes that are GC's direct ancestors (including TC). These nodes are not on the multicast tree but they know from which child they can reach the multicast tree.

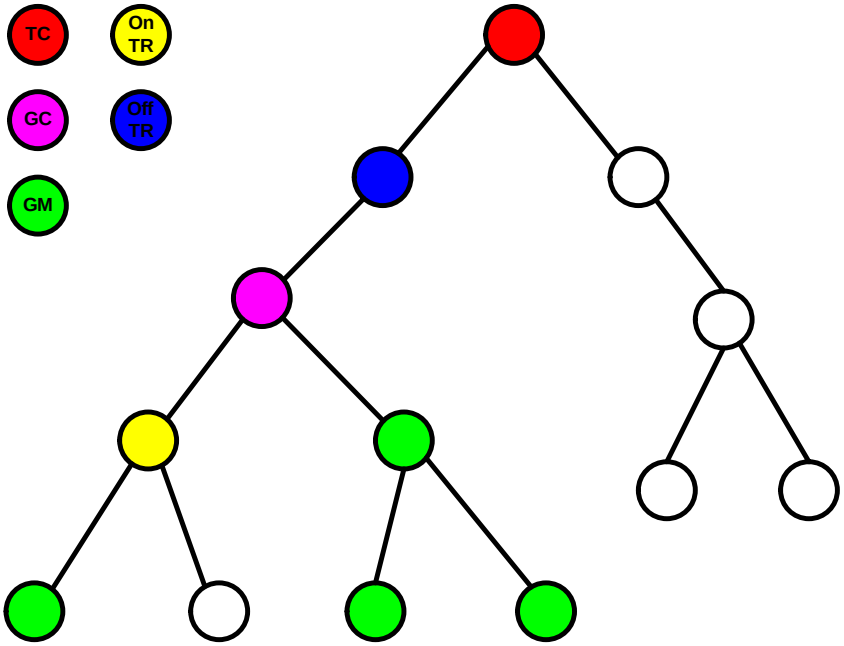

Figure 2. Entities of Multicast Protocol

2.2 Message Types

- JREQ – Joining REQuest
- JREP – Joining REPly
- LREQ – Leaving REQuest
- LREP – Leaving REPly
- GCUD – GC UpDate
- GDIS – Group DISmiss

2.3 Operation of the Protocol

2.3.1 Joining the Multicast Group

- A node wants to participate in a specific group sends (unicasts) a JREQ to its parent node if it is not an OnTR/OffTR/TC for this group;
  - If it is a OnTR for this group, it will simply change its status from OnTR to GM and its joining process is done.
  - If it is either the TC or a OffTR for this group, it simply changes its status to GC and sends a GCUD packet down to the existing multicast tree indicating it will be the new GC for this group. The joining process is then done. The current GC will give up its GC

status upon receiving the GCUD packet and dropped the GCUD packet without propagating the packet further.

- Upon receiving the JREQ from a child node, a parent node will first mark this child with the group address in its neighbor list. Then,
  - If a parent node is a GM/OnTR/OffTR, or GC, it will respond with a JREP;
  - If a parent node is none of above, it will forward the JREQ to its own parent; this process will repeat until the JREQ meets a GM/OnTR/OffTR /GC or TC.
- If a JREQ finally meets TC, it means there is not multicast branch for this node from TC. The TC will then check the record of this group address.
  - If TC found no record of this group address, then the joining node is the first one for this group. TC will respond with a JREP with the GC flag set, indicating the joining node to be the GC for this group.
  - If TC did find the record of this group address and it did have GMs of this group in its other branches, then the TC will respond with a regular JREP. Note now the TC will also be the GC for this group (it could already be).
- Upon receiving the JREP, the joining node checks whether the *GC flag* is set.
  - If it is set, then the node will set itself as the GC for this group. The joining process ends;
  - If it is not set, the route to the multicast tree is found - the node then mark its parent with the group address
- A node MAY try to join the group up to *MaxJoinAttempts* times if its previous attempts failed. If a node finally receives no JREP (even no JREPs from TC), it MAY decide to be the GC for this group.

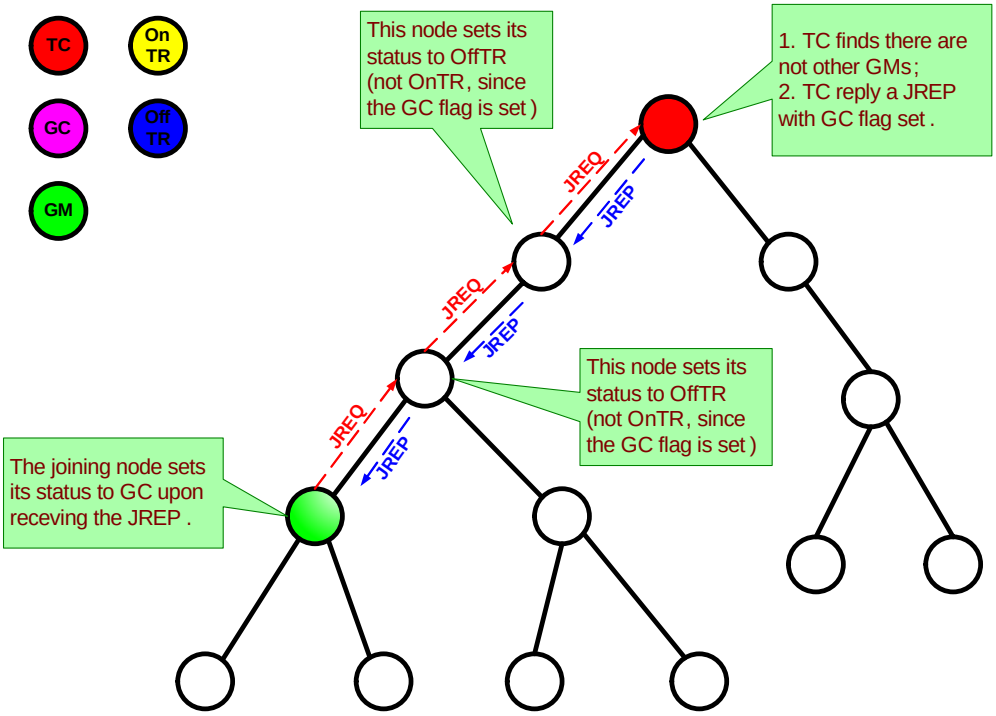

Figure 3. Joining case 1 – the first GM joins

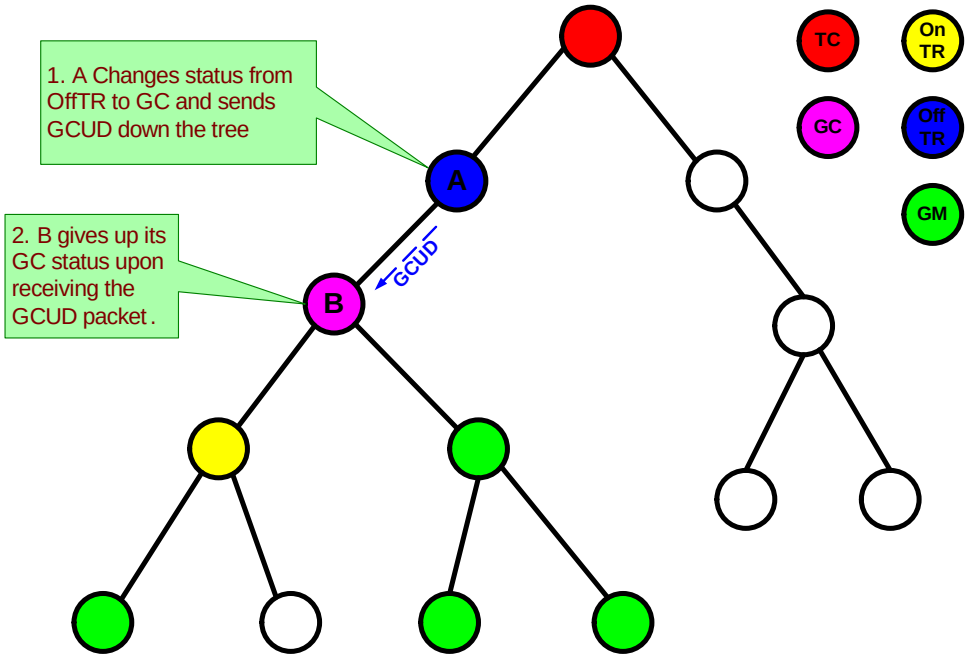

Figure 4. Joining case 2 – OffTR/GC joining

Note: This graph shows the case that OffTR joins the multicast group. TC will follow the same rule when it joins

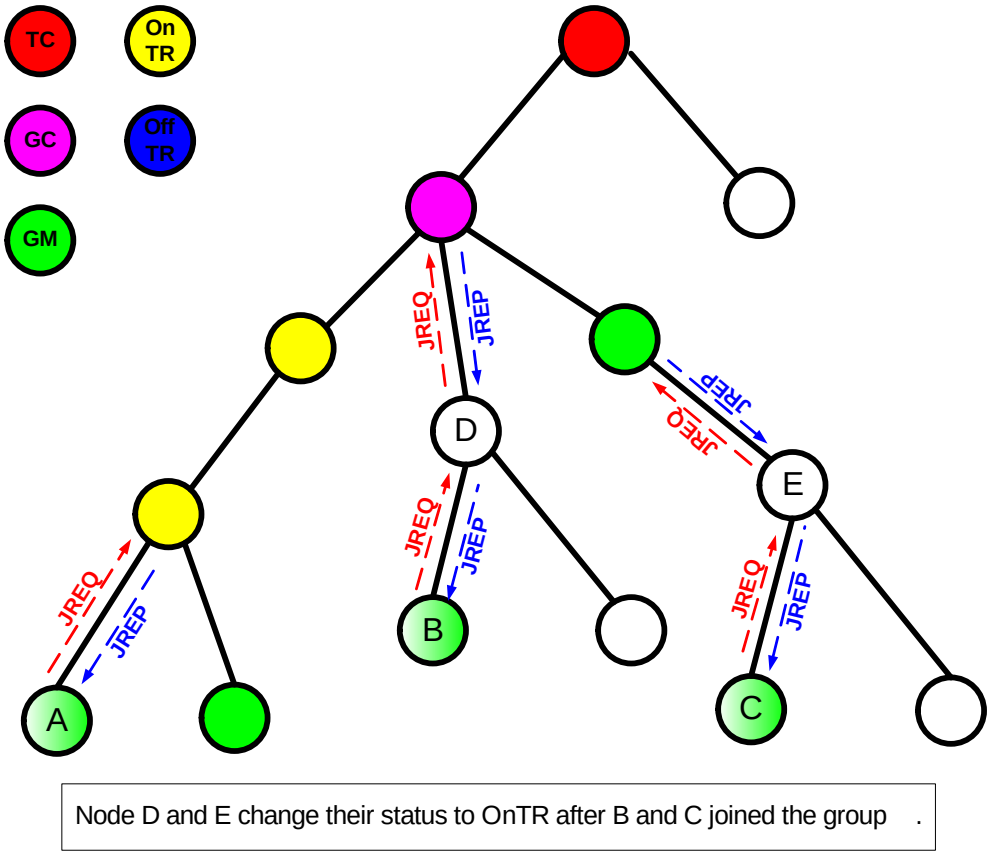

Figure 5. Regular nodes A, B and C joining

2.3.2 Leaving the Multicast Group

- A GM wants to leave a multicast group checks whether it is a leaf node. If it is, it will send a LREQ to its parent node. If it is not, it can only change its multicast status from GM to OnTR and cannot fully leave the tree.
- Upon receiving a LREQ from a child node, the parent node responds with a LREP and deletes all the multicast information related to this child.
- If the leaving of a child makes the OnTR parent node a leaf node, then the parent node will also send a LREQ to its parent to prune itself from the multicast tree.
- If the leaving of a child makes the GC have only one multicast child for a group, the GC SHOULD give up its role as the GC if it is not a GM.
- If the GC finally finds all its multicast children have left, it MAY choose to leave the group too. In this case, the GC MUST send a LREQ to all OffTRs and TC to delete all the information about this multicast group.

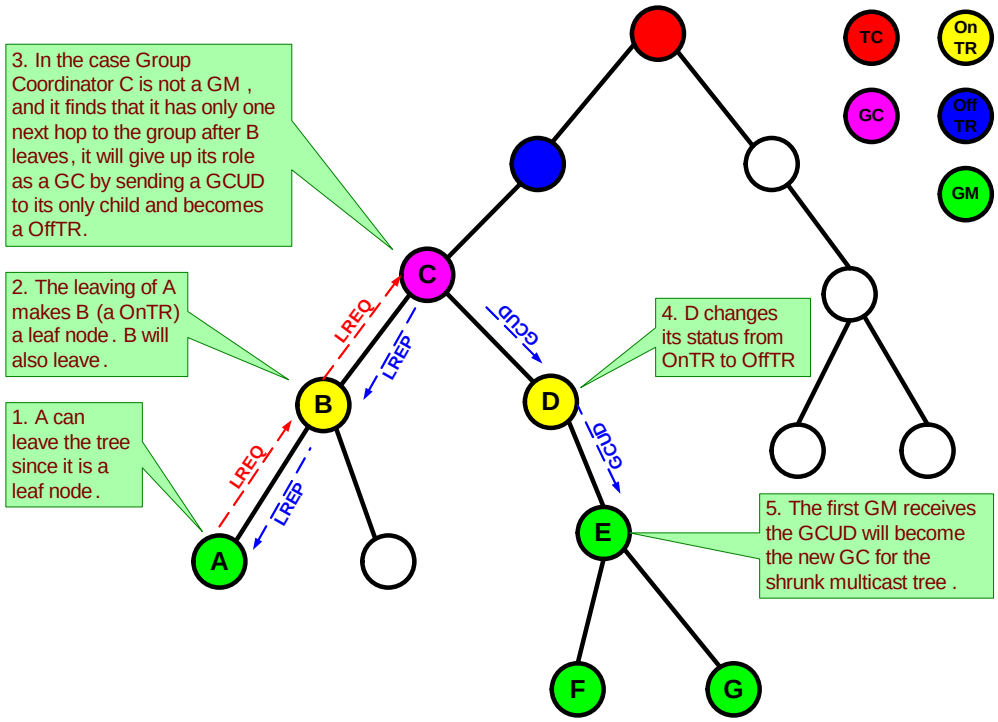

Figure 6. Leaving the multicast group

2.3.3 GC Role Migration

The GC's sole task is to set the upper bound/root of the multicast tree so that multicast packets will not propagate outside the tree. However, the upper bound of multicast tree is not a fixed one because nodes can join and leave dynamically. The role as a GC needs to be migrating from one node to another in the following two cases.

Case 1: New GM joins

When new GM joins the group, its depth of the Cluster-tree may be smaller than the current GC's depth. In this case, the role of GC SHOULD be migrated from current GC to the common parent of the newly joined GM and the existing multicast tree. The algorithm works as below:

- When a OffTR/the TC receives a JREQ from one of its child nodes (other than the existing one), it will first reply with a JREP to the corresponding child and then send a GCUD packet down to its existing child who links to the multicast tree, indicating the OffTR/TC itself is now the GC for this multicast group.
- The GCUD will propagate down to the tree until it hits the current GC.
- The current GC will give up its role as a GC upon receiving the GCUD packet. It can be a regular GM or a OnTR depending on its original status.
- All OffTRs must change their status to OnTR upon receiving the GCUD packet.

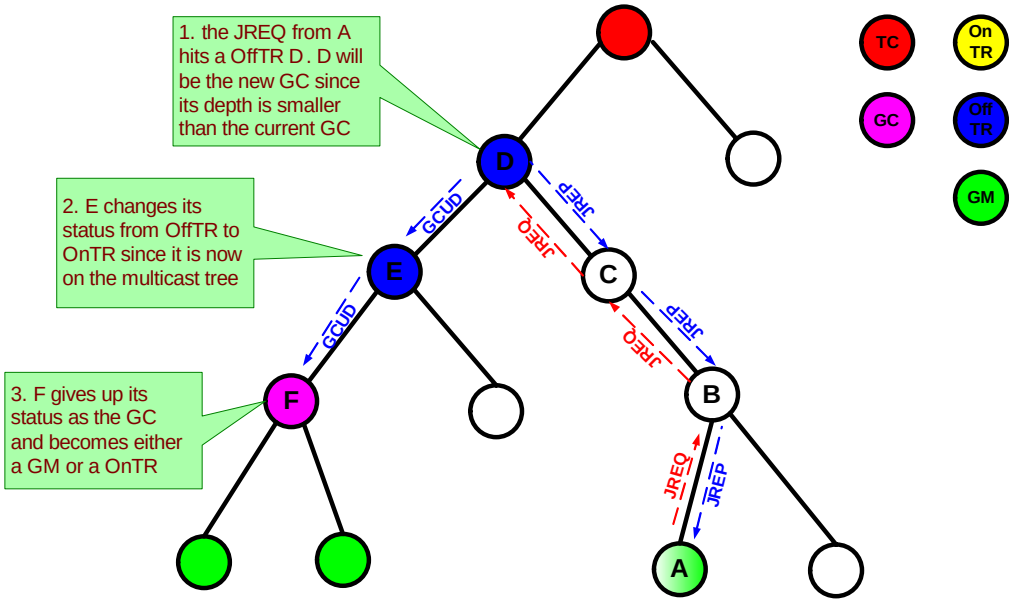

Figure 7. GC role migration when new node joins

Case 2: Current GC leaves

The current GC may leave in the following two cases:

- The GC is a GM and it wants to leave;
- The GC is not a GM, but the leaving of a child makes the GC have only one child in its neighbor list for a multicast group;

In both cases, the GC SHOULD give up its role as the GC for the multicast group. The algorithm is given below.

- When the GC detects that it has now only one child in its neighbor list for a multicast group and it is not a GM , or it is a member but wants to leave, it SHOULD change its status to OffTR and send a GCUD packet down the tree to this child (illustrated in Figure 6) .
- Upon receiving the GCUD, the first GM, or OnTR with more than one children in this group, SHOULD become the new GC for this group.
- OnTR with only one child in this group SHOULD become a OffTR instead of GC. It is now not on the tree.

Note that the GC SHOULD still be the OffTR for the group even after it leaves.

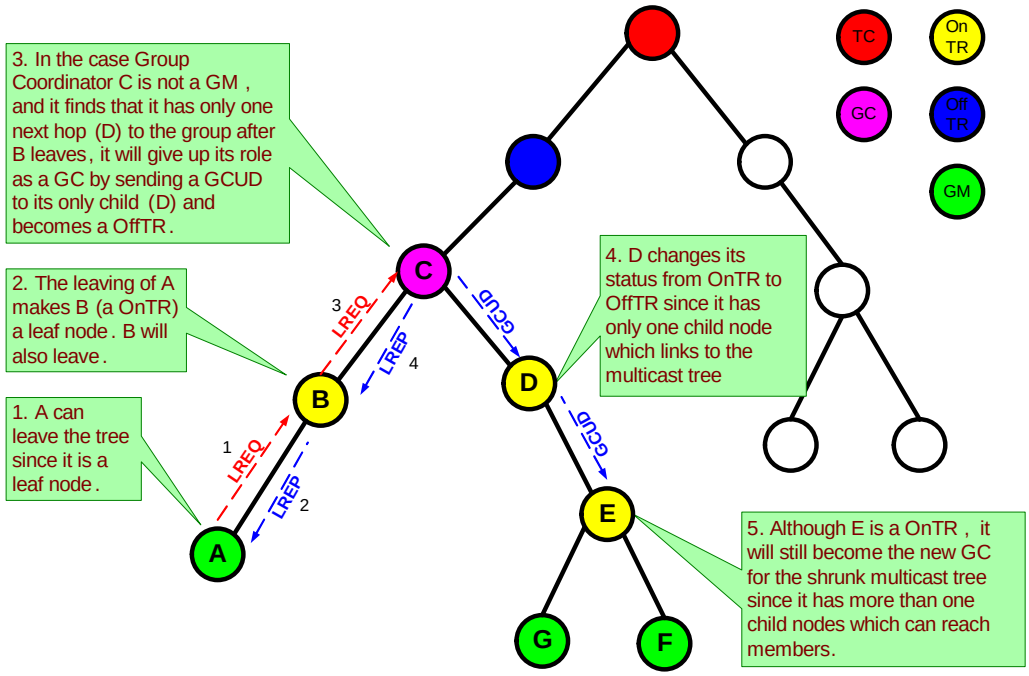

Figure 8. GC role migration when current GC leaves

2.3.4 Group Dismiss

When a group finishes its session, one of the members can issue a GDIS packet to all the members to indicate the end of the group communication. The application layer will determine which member has the right to issue this GDIS packet. Upon receiving the GDIS packet, members delete all the information related to this group. The GDIS packets reduce the control traffic led by GM's leaving process described before.

2.3.5 Data Transmission Mechanism

Upon receiving the multicast request from application layer, the Source's Mesh layer puts the multicast group address as the destination address in the Mesh layer header and indicate its MAC layer that the destination address of the MAC frame should be the broadcast address. The Source's MAC layer broadcasts the packet by setting both the destination PAN ID and MAC short address to broadcast address (*0xffff* in the case of 802.15.4).
All neighbors received this broadcast packet (at MAC layer) will pass it to their Mesh layers. By checking its own status, the "Broadcast" bit and looking at the destination address field in the Mesh layer header, each neighbor node's Mesh layer determine whether it needs to/how to process the multicast packet. The following table describes the process for neighbors with different status.

Table 1. Process for different neighbor status

Neighbor Status	Process
GM	If the previous hop of the packet is one of its on-tree neighbor (parent/child), then the GM – pass the packet to upper layer; – rebroadcast the packet if it has on-tree neighbors other than the previous hop (this check prevents leaf node from rebroadcasting). Otherwise (packets from sibling nodes), discard the packet.
OnTR	Same as GM process except passing the packet to upper layer.
OffTR/TC	If the previous hop is NOT the child who links it to the multicast tree, unicast the packet down to that child. In this case, a non-member node is sending packet to the group; Otherwise, discard the packet. In this case, the packet is out of the transmission scope. This happens when GC broadcast the packet and its direct parent (immediate OffTR) receives it.
None of above	**Unicast the packet to parent.** In this case, a non-member node is sending packet to the group and this packet can only follow the tree link toward TC. Fortunately, some packets may hit a OffTR and be propagated down to the multicast tree without going all the way up to the TC.

2.3.6 Preventing Duplicate Packets

To prevent the duplicate packet when propagating the multicast packets, a Multicast Transaction Table (MTT) is maintained in each node. Each entry of the MTT is a Multicast Transaction Record (MTR) which records the following three items of the last seen multicast packet:

- Group address (Destination address)
- Source address
- Sequence Number of the packet

2.3.7 Performance enhancement for data propagation

To reduce the processing overhead incurred by broadcast packet, a node broadcasts/rebroadcasts the multicast packet only if it has more than two neighbors on the multicast tree. Otherwise, the multicast packet is unicast at MAC layer (the network layer destination address is still the multicast group address. This can be achieved by utilizing the 802.15.4 MAC layer primitives.).

Although the basic algorithm is to broadcast packets along multicast tree, it is also possible to allow sibling GM/OnTRs to propagate the packets to expedite the packet delivery. In this case, only the Multicast Transaction Record method can be used to prevent duplicate packets.

2.3.8 Non-member Behaviors

- Non-GM can send data packets to the multicast group but cannot receive packets from the group;
- Non-GM can only UNICAST packets toward TC if it does not have link to the multicast tree;
- When a Non-GM receives a unicast packet toward a multicast destination and it is neither a OffTR nor the TC, it has to forward the packet to its parent;
- Non-GM can UNICAST packets toward the multicast tree if it is OffTR or TC because it has the information of the group;

3. Proposal Highlights

- Utilizing the underlying ART to construct shared multicast trees for different multicast groups;
- Low control overhead
  - No control traffic is broadcast;
  - In most cases, the tree coordinator is not bothered for transmitting control and data messages.
- The introduction of Group Coordinator and simple joining/leaving algorithm guarantee the multicast sub-tree is minimal at any time.
- Simple and timely data propagation.
  - Data packets do not need to go to the Tree coordinator first.

Non-members can also send packets

Key Pre-Distribution and Distributed Key Management

1. Introduction

A new generation of networks – wireless mesh networks – has been emerged recently. There are two types of mesh points (nodes) in such a network: mesh routers and mesh clients. Each mesh point may operate as a router, forwarding packets to other mesh points that may not be accessible directly due to their positioning. A mesh is dynamically self-organized and self-configured. Mesh points in the network automatically establish and maintain mesh connectivity. Without a security solution, wireless mesh network will not be able to succeed due to the lack of reliable customer oriented services. This contribution makes a proposal of using a key pre-distribution (KEDYS) for distributed key management solution to perform various security services (i.e. encryption, message authentication, etc).

2. KEDYS

KEDYS is a key pre-distribution scheme that is practical for wireless mesh networks. The scheme has the following advantages:

- The scheme is easily scalable to an arbitrary size wireless mesh network. The scalability is twofold. First, it is easy to build the scheme for a wireless mesh of any size from scratch. Second, it is easy to increase the size of an existing mesh not modifying the existing keys.
- The scheme is easy to generate. Generation complexity is the lowest possible – just proportional to the size of the mesh, and this is faster than other known schemes. Moreover, generation does not involve any complex operations.
- The process of generation of the scheme is deterministic. It does not involve any random number generators, and will be completed in a predefined time.
- The scheme is sub-optimal: the total number of keys in the scheme and those received by a mesh point are much less than in trivial and other simple solutions, and this number is smallest among known deterministically generated schemes.
- Key refresh is possible due to covering existence. Covering is a subset of keys, which belong to all nodes except the ones being excluded. Such covering has a small size and is easy to find.
- Fully distributed key management is possible due to existence of small coalitions of nodes, which are able to find coverings for the purpose of key refresh.

KEDYS is defined by the incidence matrix. It is a binary matrix whose rows correspond to keys, and columns to nodes. If in the intersection of the i-th row and j-th column there is 1, then i-th key belong to j-th node.

3. Key Pre-Distribution

In this setting, there is a Trusted Authority (TA). The TA distributes secret information among the mesh points in such a way that the specified subsets of mesh points are able to compute certain keys without the aid of the authority. The TA may go offline when the network is set up. Alternatively, the TA may stay online and render support for periodic key refresh.

4. Key Management

As a key management technique for the wireless mesh network, we propose using key pre-distribution schemes. We give generic mechanisms of using an arbitrary key pre-distribution scheme in a fully distributed setting.

4.1 Secure data exchange over mesh backbone

We assume a backbone in the mesh. The backbone is a subset of mesh points, forming a core part of the mesh. Any mesh point from the backbone is called a backbone member.
The backbone is responsible for rendering security services to other mesh points.

Mesh backbone performing security services must be protected as a whole: Any transaction, made for synchronization or as a part of a distributed protocol within the backbone, must be protected.

We assume that comparing the size of the whole mesh, the size of the mesh backbone is small. So in the backbone, we will deploy trivial key pre-distribution scheme, where every pair of backbone members has a unique secret key. Thus, if there is any transaction between two backbone members, they will use the corresponding key.

To be able to make a broadcast within the backbone, we will equip every member with a group key: A key, which is common to all backbone members. Hence any member may broadcast a message, and only backbone members will be able to retrieve the message correctly.

4.2 Mesh broadcast authentication

When a mesh point receives a message from a backbone member, he must ensure that this message indeed comes from the backbone. Therefore we need some means of authentication of the backbone's messages. To do this, we propose the use of one-time passwords in the form of a

hash-chain. Every backbone member stores the initial value for the chain (the seed) h_0. Then a cryptographic hash function $H(\cdot)$ is applied to this seed n times to compute the final hash-value:

$$h_n = \mathrm{H}(\mathrm{H}(\ldots\mathrm{H}(\mathrm{H}(h_0))\ldots))$$
$$\{\ n \text{ times}\ \}$$

This value along with the sequence number n is given to mesh points at the setup.

When a backbone member sends a message, it attaches to the message the value h_{n-1} and n. The receiver checks that $H(h_{n-1}) = h_n$. If the result is positive, he accepts the message as authentic and replaces h_n and n for h_{n-1} and $n-1$ in its storage.

In the next message the backbone will use h_{n-2}, and so on.

4.3 Secure data exchange between mesh points

All mesh points are participants of the key pre-distribution scheme, therefore they must be able to compute common pair-wise keys as the primary function using the following procedure.

Let the mesh points that wish to communicate with each other be A and B.

First, A and B exchange their columns from the incident matrix, so that they know which keys the counterpart has. This exchange can be organized in a numerous ways depending on the current topology and the network size. For example, a mesh point may store the columns of its neighbors. The mesh point may also ask the counterpart directly, or ask a backbone member. Perhaps it is easier to restore any column of the incidents matrix knowing only column's number, just as the case with KEDYS.

Next, both points make bit-wise logical AND operation over columns, the resulting column shows, which keys are common to A and B. Let these keys be $k_{i_1},\ldots,k_{i_p}$.

Then they both compute a common long-term key as:

$$k_{AB} = H(k_{i_1} \| \ldots \| k_{i_p}).$$

This key can be used directly to secure a link between A and B via a conventional encryption algorithm. Another and a more safe option is to take this key as a master secret key and using one of well-known techniques to establish a session key.

5. Distributed Key Management

5.1 Initial setup

During the initial setup phase all participants of the distributed key management system acquire all data needed to participate in the protocols in the following manner:

1. Every mesh point obtains its ID.
2. Every mesh point obtains the key block from KEDYS with a corresponding column of the incidence matrix. A member of the backbone, as any other mesh point, also obtains the key block from KEDYS.
3. Every member of the backbone obtains the corresponding key block from the trivial key pre-distribution scheme.
4. Every mesh point (except members of the backbone) obtains the final hash-value of the hash-chain, and the lengths of the chain with respect to that final value.
5. Every member of the backbone obtains the start hash-value (the seed) of the hash-chain, and the current length of the chain with respect to the final value given to the mesh points.

All information is given to the mesh point by the single logical administrative entity. This entity is conceivably some sort of setup server. After that, the administrative entity splits into two parts.

The first part, called the administrator, is placed outside of the network. Its role is then to give credentials to new mesh points. Credentials are the ID and some proof of this ID, so that the second part of the single logical administrative entity,, will be able to verify that the mesh point's ID was given to him by the administrator. The second part of the single logical administrative entity moves into the WMN itself. This may be a single dedicated server (centralized approach), or a distributed server (distributed approach). The role of this part is to give the new mesh point all other data (items 2 through 5 in the above list) basing on the ID and its proof. We assume that in distributed approach, the single logical administrative entity is the backbone. For a lager wireless mesh, the scaling of the mesh after the setup stage, when the administrative entity is split, is described later.

5.2 Key Refresh

A decision to make a key refresh is made by any member of the backbone (call him originator) for a number of reasons, for example one or two mesh points are compromised or a mesh point explicitly requested exclusion for some reason, or as scheduled key refresh. Before the refresh is started, the originator generates a session key SK .

5.2.1 Finding a covering

Having known the IDs of the mesh points being excluded, the refresh originator finds the number of keys in the covering. If the originator knows all the keys from the covering, then he proceeds with creating the refresh message or if he only knows some of the keys from the covering, he has to gather a coalition from the backbone.

5.2.2 Gathering a coalition in mesh backbone

The originator gathers the coalition in the mesh backbone in either neighbor mode or broadcast mode. The purpose is to acquire missing encrypted session key using the keys from the covering.

In the neighbor mode, the originator knows all neighbors and what keys the neighbors have. He sends the encrypted session key *SK* using the common key between him and the neighbor. The neighbor in turn replies with ciphertexts, which are *SK* enciphered under wanting keys from the covering. If the originator didn't gather all required ciphertexts (e.g. a certain member does not respond) the originator then switches to broadcast mode.

In the broadcast mode, the originator sends a broadcast request addressed to all backbone members with the goal to get session key *SK* enciphered under missing keys from the covering. The receiver then in turn replies by *SK* enciphered under the keys from the covering he possesses. The originator repeats broadcasting until all missing keys from the covering will be involved.

5.2.3 Refresh Message creating

After the gathering stage, the originator is able to create the refresh message.
This message is

$$i_1,...,i_M, E_{k_{i_1}}(SK),...,E_{k_{i_M}}(SK), c, E_{k_{SK}}(h_{c-1}), H(SK).$$

Here

$i_1,...,i_M$ are numbers of keys in the covering;

$k_{i_1},...,k_{i_M}$ are keys in the covering;

$H(\cdot)$ is a cryptographic hash function used for hash-chain and other purposes;

c is the current sequence number in the hash-chain;

h_{c-1} is a hash-value previous to the current hash-value h_c, that is possessed by the mesh points, in the hash-chain, so $H(h_{c-1}) = h_c$;

$k_{SK} = H(E_{k_{i_1}}(SK) \| ... \| E_{k_{i_M}}(SK))$ is the hash of the concatenation of all ciphertexts.

5.2.4 Refresh message propagation

The originator may broadcast the refresh message in two ways:

- Mesh point mode. In this mode, the wireless mesh is small enough so that the originator's radio transmission can cover the whole mesh.
- Mesh backbone mode. In this mode, the wireless mesh has a large coverage area and the originator cannot cover it alone and therefore will rely on other backbone members to relay the refresh message.

5.2.5 Refresh message processing

This step involves obtaining the session key *SK* and to make sure that it is authentic and comes from the backbone. Each mesh point performs the following steps:

1. Find i_q from the set $i_1,...,i_M$ such that the key k_{i_q} is in the mesh point's key block.
2. Decrypt $E_{k_{i_q}}(SK)$ obtaining a candidate session key $SK' = D_{k_{i_q}}(E_{k_{i_q}}(SK))$.
3. Check if $H(SK') = H(SK)$. If the result is positive, then $SK' = SK$ and the session key was recovered correctly.
4. Compute $H(E_{k_{i_1}}(SK) \| ... \| E_{k_{i_M}}(SK))$ and decrypts $E_{k_{SK}}(h_{c-1})$ obtaining a candidate hash-value $h'_{c-1} = D_{H(E_{k_{i_1}}(SK)\| ... \| E_{k_{i_M}}(SK))}(E_{k_{SK}}(h_{c-1}))$.
5. Check if $H(h'_{c-1}) = h_c$, where h_c is the hash value stored by the mesh point. If the result is positive, then the mesh point is assured that renewal message is authentic and comes from the backbone. The mesh node then updates h_c with the new $h'_{c-1} = h_{c-1}$.

5.2.6 Key block refresh

The key block refresh is as follows:

A new value of the i-th key is k'$_i$ = H (k_i||SK)

6. Mesh Scaling

After the setup stage, the size of the wireless mesh is changing. Some new devices become mesh points; some mesh points quit the network. Key pre-distribution scheme address these issues as follows.

6.1 Mesh point exclusion

When a mesh point is excluded from participating in security services from other mesh points, some backbone member starts the key renewal procedure, and all the backbone members marks the column of the incidence matrix corresponding to the excluded mesh points as excluded.

6.2 Mesh point association

When a new device wises to join the wireless mesh, he starts the procedure of new mesh point association.

The first step for a potential mesh node is to contact the administrator (the first part of the single logical administrative entity) and to obtain ID and proof for the ID.

The potential mesh point contacts any member of the backbone through a location-limited channel (e.g. one touch port, wireless channel with extremely small coverage, etc.). When the potential mesh point contacts the backbone member, this member checks the ID and the proof of it. Then he generates the current hash-value from this hash-chain and fetches from the member's key block keys that will be common with the potential client. The whole bundle (the column of the incidence matrix corresponding to the ID, current hash value and sequence number, the keys) is passed to the potential node. The node then contacts another member of the same backbone, and this member does the same as the previous one. The potential mesh node contacts backbone members until it gathers the whole keys block. Due to properties of KEDYS the number of members to contact is small enough.

6.3 Lost mesh points recovery

When a mesh point is lost due to any number of reasons (e.g. mis-synchronization, power failure, etc), the mesh point contacts a member of the backbone through location-limited channel. The member checks if this client was marked as excluded. If it was, the member refused to recover the node. If it is not, the member fetches from its cache the session keys that were used for key refresh since the node was lost. Using these session keys, the node makes several key refreshments and finally obtains a key block, which contains the keys consistent with the current key pool.

If the node missed too many key refreshments it will start the procedure of the mesh point association as if this node were a new node.

7. Clustering

When the network is sufficiently large, the concept of clustering can be used to logically divide the network into smaller sub-networks, or clusters. In every logical cluster, a separate KPDS is established using the identical incidence matrices but different keys. The clusters must be connected by some kind of bridge. The clusters may be organized in different security perimeters so each cluster has its own perimeter. In addition, different clusters may be organized within a single security perimeter.