|  |  |
| --- | --- |
| Zigbee Network Layer Tutorial - Part 1: The Tx Data Path |  |

|  |
| --- |
| <http://freaklabs.org/index.php/Blog/Zigbee/Zigbee-Network-Layer-Tutorial-Part-1.html>  Written by Akiba |
| Saturday, 09 May 2009 |
| I've been meaning to get around to this for a while now, but things have been so busy that I just haven't had the time. Or maybe I was just too lazy. In any case, today is a good time to kick off a series detailing the Zigbee NWK layer, mostly because I caught a cold and it pretty much rendered me useless for any heavy thinking.  I know it's a bit late to start a series on the Zigbee NWK layer after they announced that they were transitioning to IP. However, I think that it'd be a good study on how Zigbee stands right now, where you'll be able to see the strengths and weaknesses of the networking protocol. Hopefully, you'll also be able to see what role IP will play in it, and what areas are and aren't a good fit for IP.  So here we go…  ***The Zigbee 2007 Networking Layer - Part 1***  The Zigbee networking layer is probably one of the most complex layers in the protocol stack. Mostly it's because they spent a lot of time outlining all types of different conditions, routing methods, a bunch of tables, etc since it's the backbone of the transport part of the stack. Since it's quite a large topic, I think the best way to go is to divide it up into parts and dissect each part individually.  The first cut will be to split the layer into two sections: the data path and the management path. Both sides are fairly complex, but if you understand the data path well, then you can probably see why the management side is also complicated.  Below, you can see a very simple diagram of how the network layer looks from a data point of view. Data to be transmitted can either flow down from an upper layer, received data can come up, and data can do a U-turn where received data comes up, gets analyzed, and then gets retransmitted (forwarded) to its next hop destination.  igbee Simple Network  On the transmit side, the Application Sub Layer (APS) sends data down to the network layer via a data request. Inside the data request, the APS specifies the destination address, whether or not route discovery will be allowed, and the radius. The radius is the maximum hops that the frame will be allowed to travel. This is mostly used in broadcasts where you might not want the broadcast to travel too far, or it's usually set to a default value to prevent endless hopping in case there's a loop in the routing tables.  Once inside the data request, the network header is filled out and then the address is examined. At this point, it's probably best to make a distinction between network addressing and MAC addressing. There are two sets of addresses in each Zigbee frame: the 802.15.4 source/destination addresses and the Zigbee network source/destination addresses. Actually, sometimes only the source or the destination is present depending on the situation, but that's another article.  The key to understanding the Zigbee routing algorithm is to understand how both addresses are used. The network source or destination address is what I'll call the absolute source or destination. This means that if a frame originated from address 0x1234, this will be the source address in the networking field no matter what. The same goes for the networking destination. On the other hand, the MAC address is used as a next hop address which could just be a stepping stone on the way to the destination. It's like if I wanted to go to Akihabara. I know where I came from, and where I want to go, but from a microscopic perspective, I have a couple of interim destinations (train platforms) I need to get to before I reach Akihabara. And one more thing about addressing, for the purpose of this series, I'm only going to be referring to the 16-bit short addresses used by the MAC layer. The MAC actually has a 64-bit address as well, but it would just confuse things.  igbee Addressing  Okay, so now where was I…ahhh yes, how the network data request processes the addresses. Inside the network layer on the transmit side, you only have one piece of real information which is the destination address. From that you'll have to figure out the next hop address for the MAC. You can say that most of the network layer exists just to figure out what the next hop address will be.  To figure out the next hop, you have to go through a series of comparisons. Although the order is dependent on the stack, I'll refer to the order that I use. The first comparison is to see if the destination is a broadcast. Zigbee has a few classes of broadcasts: broadcast to everyone, broadcast to routers, and broadcast to non-sleeping nodes. Upon checking the spec again, I see that there is a broadcast for low power routers as well, although I' haven't seen this used in the spec.  I'll go into broadcasting in more detail later, but suffice it to say that if the destination address falls into the broadcast category, then the next hop address is simple…it will be the 802.15.4 universal broadcast address which is 0xFFFF. Case closed, ship that frame out to the MAC…errr…kind of. You'll see later on that broadcasting isn't that simple. Precautions need to be taken because you because you can easily crash a network with a poor broadcast.  Moving right along, if the destination address isn't a broadcast, then you need to determine how to get it to where it's supposed to go. I check the neighbor table first to see if the address matches any destination inside of it. The neighbor table, as you might imagine, is a list of all the neighbors that are within earshot of the node. I'll describe this in more detail later on as well, however this is the next logical place to check for the next hop. If the address is inside your neighbor table, then you can get the frame to its destination in one hop and don't have to go through all the fancy, shmancy routing algorithms.  If it's not inside your neighbor table, then you have to resort to forwarding it to an intermediate node on its way to the final destination and this is where things might get a little complicated. I check my routing tables first because if the destination address is inside my routing tables, the routing table will include the next hop address and I can just send it out. However if it's not inside the routing tables, then you have to decide if you want to do a route discovery or if you want to route it along the tree (my current stack supports tree routing as well...this won't be the case if you have a Zigbee Pro implementation which only supports the mesh routing).  The tree routing is the last resort for me so the next thing I do if I can't find the destination is to check to see if the frame will allow route discovery. Route discovery takes a bit of time, and also floods the network with broadcast frames so there may be some instances that you might not want to do this. However if the route discovery flag is true, then I buffer the frame in a holding queue and kick off a route discovery to find the destination within the network. For more information on this process, check out the article [I wrote a long, long time ago on Zigbee mesh routing](http://freaklabs.org/index.php/Articles/Zigbee/zigbee-mesh-routing-interactive-tutorial.html" \t "_blank) .  And finally, if I can't do a route discovery, then I just route it along the tree, using the algorithm outlined in the Zigbee spec. Again, for more information about tree routing, as well as why it sucks, check out [this article I wrote on it a few months ago](http://freaklabs.org/index.php/Articles/Zigbee/Zigbee-Tree-Routing-How-It-Works-and-Why-It-Sucks.html" \t "_blank) .  If all hell breaks loose and you can't do route discovery or tree routing (your upper layers are sadistic, pointy-haired bosses), then you just give up and return an error status in the confirmation.  Whew…that was quite an introduction to just the transmit side of the Zigbee network layer. Join me next time as I discuss the ever exciting receive side data path which is pretty much the same, but with an added twist. |

|  |  |
| --- | --- |
| Zigbee Network Layer Tutorial - Part 2: The Rx Data Path |  |

|  |
| --- |
| Written by Akiba |
| Tuesday, 12 May 2009 |
| I left off last time explaining the transmit data path side of the Zigbee networking layer. The receive data path is fairly similar, but there are some minor complications.  When a frame arrives over the air, the radio driver will take it out of the buffer and store it somewhere. It should then signal the next higher layer (in this case the MAC) to retrieve the frame.  Incidentally, this part of incoming data handling is common to just about every protocol stack. I refer to it as the “launch” although I don’t know what the formal term is. I call it the launch because it’s the entry point of data into your stack. In general, most stacks that I’ve encountered handle things the same way where the hardware signals data arrival with an interrupt, the data is read out of the hardware buffers by the driver, it’s put into a holding area (list, queue, circular buffer),  and then the next layer is signaled to pick it up when it can. This way, the ISR is kept short and frame dropping is minimized in the case of heavy traffic.  Anyways, back to the story. After the launch, the MAC layer would pick the frame up from its holding area, strip off and decode the MAC header, and then if it’s a MAC data frame, pass it up to the NWK layer.  The frame reaches the NWK layer via the MAC’s “Data Indication” service. Once in the network layer, the network header will get stripped off and processed. This is kind of the generic way to handle an incoming Zigbee frame and I think most stacks will do something fairly similar. From here, it pretty much depends on the implementation so I’ll describe how I handle the network frame processing in the FreakZ stack.  After I have the network header parsed and tucked away neatly inside a structure, the first thing that happens is that the frame needs to be decoded according to its type. There are only two frame types at the network layer: data frames and command frames. The frame type is contained in the header information so it’s easy to figure out what it is. In the case of a data frame, there are three cases that need to be handled:  The first case is the easiest one where the frame is meant for us. To determine this, you need to check the destination network address and see if it matches the device’s network address. As I mentioned in part 1, the destination address in the network header is absolute. So if the destination network address matches our network address, then we can be sure that we’re the final destination for this frame and just send it up to the next layer.  If the destination address doesn’t match our address, then the next thing I do is check to see if it’s a broadcast. If it’s a broadcast, then we need to initiate a special sequence of events because broadcasts need to be handled carefully. The dangerous thing about them is that broadcasts can grow exponentially if they’re not handled properly and they’ll quickly overwhelm the memory of your devices, causing your network to crash. I’ll be discussing broadcasts in more detail in the later parts where I examine each of the network services. Also, since a broadcast hop is still considered a hop, we need to decrement the frame's radius value. If you remember from part 1, the radius is used to limit the max number of hops a frame can travel. Basically any time a frame needs to be re-transmitted, the radius value will be decremented.  And finally, if we’re not the final destination for the frame, and it’s not a broadcast frame, that means that we’re just a hop on it’s way to the final destination. From there, the sequence of events is similar to the transmit data path discussed in part 1. We need to find the next hop address to send the frame, forward it there, and decrement the radius counter.  Since we have a commonality like this, I made a simple optimization where both the transmit and the receive side share the same forwarding function.  igbee Rx Data Path  In the case that the incoming frame is a command frame, then I just decode it based on the command ID. This will always be the first byte of the payload so its fairly easy to get this value. From there, it’s just handled according to the management function that it needs to accomplish. The command frames in the network layer handle common network maintenance tasks like mesh route discovery, link maintenance, and rejoining and leaving the network. I’ll be going into this in more detail later on as well. |

|  |  |
| --- | --- |
| Zigbee Network Layer Tutorial - Part 3: Broadcasts and Neighbors |  |

|  |
| --- |
| Written by Akiba |
| Monday, 18 May 2009 |
| I thought I would spend this portion of the series discussing some of the more detailed parts of the data path. Since the [tree](http://freaklabs.org/index.php/Articles/Zigbee/Zigbee-Tree-Routing-How-It-Works-and-Why-It-Sucks.html" \t "_blank) (see below) and [mesh](http://freaklabs.org/index.php/Articles/Zigbee/zigbee-mesh-routing-interactive-tutorial.html" \t "_blank) (see below) routing is already explained in other articles, I’d like to talk about the other two forwarding methods: broadcasting and the neighbor table.  For those new to the Zigbee spec and trying to implement the broadcast functionality, it can be pretty confusing. At least it was for me when I had to figure it out so hopefully this can help clear some of the haze. Broadcasting plays an important role in Zigbee and is used for many functions. Two of the most prominent are route discovery and group transmissions. Route discovery is the process of locating a path to a destination address whose route is unknown. Zigbee uses a modified form of AODV (Ad-hoc On-demand Distance Vector) which is just fancy terminology for “flood the network with pings until you hit the destination address”. The flooding part occurs by broadcasting route requests and have them propagate through the network until the destination is reached. Group transmissions are a method of transmitting data to all devices within a certain group. A broadcast is used to transmit the data and the frame will be discarded by any members that don’t belong to the group. Along with those functions, there are numerous other smaller functions that utilize broadcasts in both the ZDO (Zigbee Device Object or endpoint 0 on all Zigbee devices) and the ZCL (Zigbee Cluster Library).   To understand broadcasts, it might make sense to discuss some of the different device types a bit. There are three types of Zigbee devices: the coordinator, routers, and end devices. The coordinator is just a router that starts the network. It always has a network address of 0 and mainly performs the function of scanning the network and selecting the channel and ID for the network. A router is a device that has the capability to forward frames and usually, is able to accept child devices. An unfortunate attribute of routers and coordinators in Zigbee is that they’re unable to sleep which is a standard complaint among many people that are investigating using Zigbee for wireless sensor networking. This limitation means that Zigbee routers usually need to be attached to a MAINs power supply.  An end device has no resources to forward frames and can only join and communicate with a parent router. The simplified communication capabilities allow most of the MAC, NWK, and APS management functions to be stripped out and should result in a very small memory footprint. Sleepy end devices are able to be duty-cycled where they sleep most of the time and awaken periodically to poll its parent for any buffered messages. It uses 802.15.4 indirect transmission for the polling, which is [discussed in more detail in my 802.15.4 series](http://freaklabs.org/index.php/Articles/Zigbee/IEEE-802.15.4-in-the-Context-of-Zigbee-Part-2-MAC-Layer.html" \t "_blank). Duty cycling the end device allows it to consume very little power, thus increasing the battery life which is one of the most important factors in wireless sensor networking.  So anyways, back to broadcasting. The reason I discussed the device types is because you can target your broadcasts based on the device type. There are four broadcast addresses that can be used depending on your broadcast audience:   |  |  | | --- | --- | | **Address** | **Audience** | | 0xFFFF | All Devices | | 0xFFFD | All Devices with Receiver on Permanently | | 0xFFFC | Routers and Coordinators | | 0xFFFB | Low Power Routers |   Just a note, although low power routers are specified, I haven’t heard of any actual implementations of them yet. Feel free to correct me on this.  Transmitting a network broadcast frame in Zigbee actually sets off a chain of events. If a new broadcast is received, either from another device or from a higher layer, a broadcast transaction record is created. If the frame was received from another device, a copy of the frame is also made and sent up to the next layer for processing.  The broadcast transaction record is used to track the source address and sequence number of the broadcast. These two pieces of information are used to uniquely identify a broadcast frame. This is important because once the broadcast frame is forwarded, all neighbors within earshot will re-send the broadcast frame and you’ll get multiple copies of it. As long as you have the broadcast transaction record, you’ll know that you’ve already received and processed the frame so you can discard the copies.  **Broadcast Transaction Record** **Entry**:   |  |  | | --- | --- | | **Field** | **Description** | | Src Address | 16-bit Network Address of Broadcast Initiator | | Sequence Number | Network Layer Frame Sequence Number | | Expiration Time | Amount of Time Before This Entry Expires |   The record that was created actually goes into a table called the Broadcast Transaction Table, or BTT. The BTT implements what’s called a passive acknowledgement system, and is used to ensure that all known neighbors have received the broadcast sent by the device. As I mentioned previously, when a broadcast is transmitted, all devices that receive it will broadcast a copy. Each time a copy of the broadcast arrives, the address of the sender will be added to the BTT to mark that it has relayed the broadcast. After a broadcast timeout, if all neighbors haven’t relayed the broadcast, meaning they aren’t present in the BTT, then the original sender will need to do a broadcast retry. This happens until the max retries (usually 3) or all the neighbors show up in the BTT.  Apologies if it might sound a bit confusing. I had to read the broadcast section more than a couple of times and actually wrote some simple simulation programs to gain an understanding of the behavior and the passive ACK mechanism.   The problem with the BTT is that it is not very deterministic. If there are a lot of neighbors, ie: the network is dense, then the broadcast transaction table has the potential to become large, eating into the RAM. Thus there is an option in the Zigbee specification to forgo the broadcast transaction table. The tradeoff is that the device will need to broadcast the frame the maximum amount of retries for any broadcast. At first glance, this would be desirable because you can get rid of the BTT which has an unknown number of entries. However this also means that each broadcast will be retried three times (the default retry number), taking a toll on all devices on the network. Each received frame, whether a duplicate or not, requires RAM since it needs to get to the network layer before it can be checked and discarded. If many devices on the network have no broadcast table, then each broadcast would generate a huge amount of traffic, possibly triggering some devices to run out of memory. So when you see warnings in the software documentation that broadcast transmissions should be used sparingly, they’re usually referring to the fact that a broadcast storm may chew up the available RAM in a node.  Since the mesh and tree routing mechanisms as well as the broadcasts have been covered, it's time to discuss the final method of data forwarding, which is the neighbor table. The neighbor table contains a list of the devices that are within transmission/reception range and provide a convenient single hop transmission to the destination. It is also used during the discovery or rejoin process to see if the joining device was previously a child of the node. According to the specification, the neighbor structure contains both mandatory and optional fields. However in actual usage, the optional fields are required since they will be needed by some of the ZDO functions. Just a little gotcha for those implementing their own stack:  **Neighbor Table Entry - Mandatory**   |  |  | | --- | --- | | **Field** | **Description** | | Extended Address | 64-bit device address | | Network Address | 16-bit network address | | Device Type | Coordinator, router, end device | | Rx On When Idle | Flag to mark sleepy end devices | | Relationship | Parent, child, sibling, no relationship | | Transmit Failure | Transmission failure counter | | LQI | Link quality indicator | | Outgoing Cost | Cost of outgoing link as measured by neighbor. *Only required if symmetrical links are used.* | | Age | Time since link status command was received. *Only required if symmetrical links are used.* |   **Neighbor Table Entry - Optional (Not)**   |  |  | | --- | --- | | **Field** | **Description** | | Extended PAN ID | 64-bit unique PAN ID | | Channel | Operating channel | | Depth | Tree depth of device | | Beacon Order | 802.15.4 Beacon Order. Zigbee uses no beacons so this should be 0x0F | | Permit Joining | Flag to indicate whether device is accepting join requests | | Potential Parent | Flag to indicate if neighbor satisfies parent criteria |   The neighbor table is initially populated during device discovery when a device is searching for a parent to join to get on the network. I’ll discuss the join procedure later when I get into the network management side of things. Anyways, when a device tries to join a network, it will first perform a device discovery where it broadcasts a beacon request. All routers within earshot will respond with a beacon frame containing information about themselves. This information will get stored into the neighbor table. Unfortunately, the spec is a bit light on details about populating the neighbor table after the initial join procedure. In order to keep the table up-to-date, any beacons that are seen should be compared to the neighbor table and added to it if an entry doesn't exist.  Well, that kind of takes care of most of the main points of the network layer’s data path. Within the Zigbee stack, or even the full protocol stack including 802.15.4, I’d say that the network layer data path is the most complex. Next up should be the network management which includes device discovery, joining, leaving, and network maintenance.   |  |  | | --- | --- | | Zigbee Network Layer Tutorial - Part 4: Network Management 1 |  |  |  | | --- | | Written by Akiba | | Thursday, 04 June 2009 | | Welcome back to my ongoing series of Zigbee tutorials on the soon-to-be obsolete Zigbee Networking layer. Now that we’ve finished covering the data path, it’s time to look at the network management services. It’s quite a wide topic so I’ll start with the main management services and then cover some of the more esoteric ones later. Before we begin, I think it might be good to give a quick peek ahead on how a user application would interface to the networking layer to manage the network. With a top-down view, things might make more sense than just listening to me talk about some abstract next higher layers that all the data magically goes to or comes from.  ***Brief Overview***  Whereas the Zigbee networking data path logically follows the protocol stack layer hierarchy, the network management deviates somewhat from it. Since this may be a bit confusing, let me try to explain it briefly.   The application layer consists of three parts: the Application Sublayer (APS), the Application Framework (AF), and the endpoints. The Application Sublayer interfaces the Zigbee application layer to the Zigbee networking layer and it provides a common set of data transport services to all the endpoints. There are also a couple of other services that the APS provides and we’ll get into that when we discuss the app layer in more detail.   The Application Framework is just a glorified multiplexer and container for all of the endpoints. All of the endpoints register themselves with the AF, and when a data frame comes into the application layer, the AF will check its destination endpoint and forward it there.   The endpoints are what most people associate with Zigbee. Each endpoint houses what’s called an application object which is basically a device profile with whatever extra functionality you decide to add. When the device is started, all the endpoints will register themselves with the application framework and provide descriptions of their device profile and their capabilities. Endpoint 0 is a special endpoint and always contains the Zigbee Device Object (ZDO). This object implements the Zigbee Device Profile which has multiple functions, one of them being the network manager.   The user application can manage the network by making requests and handling callbacks to this object, which is why it’s important to know about it. In general, the Zigbee endpoints are going to be the main interface between the user application and the stack.   When I mentioned that the network management deviates slightly from the stack’s layer hierarchy, what I meant was the ZDO’s network manager object bypasses the APS and interfaces directly to the networking layer. Thus, you get a bizarre looking layer stackup like this where you can see the ZDO wraps around the APS and the NWK layers.  igbee Architecture  *(Graphic gratuitously copied from from Zigbee Open House - Seoul, 2006 - Phil Jamieson)*  You can also forget about all of those SAPs (service access points) between the layers, since they’re mostly there for formality. In a real implementation, you would just call the functions directly or via function pointers.   With that out of the way, we can take a look at a few of the main network management functions.  ***Network Formation***  When the user app decides to form a network instead of joining an existing one, it will instruct the ZDO to call the network formation function. Only a router that is coordinator-capable can form a network and this is indicated in the application layer’s information base. It’s just a fancy term for the app layer’s configuration table.    When the network formation function is called, a list of allowed channels needs to be supplied to which may be limited to a subset of the total available channels (16 channels @ 2.4GHz). It’s usually based on certain requirements like avoiding channels that overlap with an existing 802.11 network. Incidentally, coexistence of Zigbee with 802.11 is a major debate point between other competing radio providers. The main issue is that they claim Zigbee is susceptible to interference from Wi-Fi networks whereas other radios operating at non-2.4 GHz frequencies are not. I won’t inject myself into the debate here, but I’d suggest locating any routing nodes a few feet away from an 802.11 access point. I think 3-6 feet is the magic number that I often hear. Otherwise, just take care to avoid 802.11 channels in your channel list.  The network formation function will call the MAC’s energy scan and active scan services and perform scans on the supplied channel list. When the scans are finished, the MAC’s scan confirm function will return the energy readings and network scan descriptors to the function via the MAC’s scan confirmation. From there, the network formation function will need to decide on the channel to join. The usual criteria is to choose a channel with the lowest energy reading (lowest amount of traffic) and the fewest networks. If you have access to the source code, you can also modify the channel decision function to add additional criteria.  Once the channel is decided on, the newly crowned coordinator will decide on a PAN ID and set the channel in the radio. The final step is for the NWK layer to call the MAC start service which configures the MAC layer. After that, confirmations go back all the way up to the user app.  etwork Formation Sequence  ***Network Discovery***  As the name implies, the Zigbee network discovery service is used to discover the existing networks on the current channel. It’s mostly just used when the device is started to find out if there are any suitable networks to join, although it can also be called at any time via the user app.  When a network discovery is requested by the ZDO (or user app), the discovery function will call the MAC’s active scan service which, in turn, will broadcast a beacon request. When other devices see the beacon request, they will respond with an 802.15.4 beacon frame. If you remember from my 802.15.4 articles, an 802.15.4 beacon frame contains MAC information about the responding device as well as a beacon payload for generic data. Within that payload, the responding device will include Zigbee network information such as the protocol ID and version, amount of routers and end devices allowed to join, the device profile that is being used, and other somewhat useful information.  eacon Network Payload  When the beacons from the scan request are received, the device will add both the MAC and NWK info to its scan descriptor list and its neighbor table. After all of the beacons have been collected, a network discovery confirmation will be sent to the ZDO along with the list containing all the scan descriptors. The ZDO or the user app would then need to decide which network to join based on certain join critera. It’s here that the user can specify if they only want their device to join certain networks or even if there is a specific device they’d like to join to.  etwork Discovery Sequence  ***Network Join***  Joining a device or allowing a device to join is probably one of the most complicated processes in Zigbee. There are actually two sides to the network join function: the child side which sends the request and the parent side which processes the request and sends the response. To get a thorough understanding of the join process, I’ll treat each sequence separately. I won’t go into the details of the 802.15.4 association sequence since an explanation of that can be found in the MAC tutorial.   ***Network Join – Child***  The first part of the join process for the child is to do a network discovery. This is usually done when the device is first started and is not associated with any network as mentioned previously. Once the network discovery is finished and the potential parent has been decided on according to the join criteria, then it’s time for the network join process to start.   When the potential parent has been chosen, a network join request is called by the ZDO. The network join request will call the MAC’s association service and issue an association request to the potential parent. From there, the procedure follows the MAC’s association sequence until the association response is received from the potential parent.   When this response is received, it will get passed up to the network layer via the MAC’s association response. If the join was successful, the device will update it’s NWK and MAC information tables to include the new network address, PAN ID, and also update the neighbor table to specify its parent. Once the administrative work is taken care of, the network join confirmation is sent up to the ZDO where it can inform the application about the join status. If the join status was unsuccessful, then the ZDO/user app will choose another potential parent from the neighbor table and retry the join procedure until it eventually joins a network or runs out of potential parents.   One of the last things that occur after a successful join is that the device will broadcast a device announcement informing everyone on the network that it has joined the network as well as it’s 16-bit network address and 64-bit IEEE address. This is important because if the device was previously joined to the network with a different network address, the other devices will be able to find out from it’s IEEE address and can clear all references to the old network address. Also, the address info will be added to everyone’s address map which tracks all the devices on the network.  igbee Join Sequence - Child  ***Network Join – Paren***  The parent side of the join process is slightly easier. When a MAC association request arrives at the potential parent, it sends an indication to the network layer that a device is trying to join. The potential parent will then search its neighbor table to see if the 64-bit IEEE address already exists. If it does, then that means that the device was already previously joined and the parent will just issue the same network address to it. If not, and the parent is allowing devices to join it, then it will simply add the device to its neighbor table specifying that it’s a child device and generate a new network address for it. This all gets packaged up and sent out as a MAC association response. Again the rest goes according to the MAC’s association service.  igbee Join Sequence - Parent    Well, I guess that just about takes care of this part of the network management functions. Tune in for another chunk of network management functions in the next part. | |

|  |  |
| --- | --- |
| Zigbee Tree Routing - How It Works and Why It Sucks |  |

|  |
| --- |
| Written by Akiba |
| Sunday, 11 January 2009 |
| Zigbee 2007 (Residential) has two methods of routing: mesh routing and tree routing. Although most people talk about the mesh routing capability of Zigbee, not too many people know much about the tree routing. There's actually a good reason that it isn't discussed much and I'll get into that later. However I do receive the occasional question on how Zigbee tree routing works. Mostly, its from people that want to run Zigbee using the 802.15.4 beacon mode since tree routing is the only routing method supported in this case.  In order to understand tree routing, its first important to understand the underlying principle which is how addresses are allocated. Once that is understood, the routing becomes trivial.  **Tree Address Allocation**  Tree routing works by routing frames based on the addresses of the routers in the network. In order to do this, the routers that join need to have their addresses allocated in a special way. Although the address allocation scheme can be summarized in a handy little algorithm called CSkip, which can be found in the Zigbee spec, it's probably best to explain qualitatively how it works.  Actually, the allocation scheme isn't very sophisticated. You start off with an address space that is calculated based on the maximum number of routers per device (Rm), max children (routers + end devices) per device (Cm), and max depth (Dm) of the network. To make things simple, lets just take the case where the "max children = max routers" which means that all the children will be routers and no end devices are allowed on the network. The simplification just makes it easier to calculate the address space.    Based on the information above, we can calculate the size of the address space, since it will be the same as the maximum number of devices on the network. In other words, each device gets one address. As an example, let's say that the maximum number of routers that can be joined to a device is two, the maximum number of children for a device is 2 (hence all children are routers), and the max depth of the network is two. In Zigbee speak, this would be labeled (Rm=2, Cm=2, Dm=2).  This would give us a network that looks like this:  asic Tree Network  From the picture, you can see that the maximum number of devices would be 7, hence the address space will span from 0 to 6. Also I purposely drew it in a tree structure (in this case binary tree) for purposes that you can see below.   The next issue is how to divide up the addresses between the devices. Although it makes sense to allocate the addresses based on a first-come first-serve strategy, that wouldn't give us any routing information. Thus the addresses are given out based on a tree hierarchy, which incidentally, is where the name comes from. Here is how the addresses would look for each device:  asic Tree Network with Addresses  All of these addresses are fixed for the position in the tree that a device occupies. You can see that the numbers progress sequentially from top to bottom and left to right as you follow the flow of the tree. Hence the 1st address which corresponds to the first device on the network will be 0. This corresponds to the coordinator and the coordinator's address will always be zero in a tree address allocation scheme. The first device that joins the coordinator will have an address of 1 and it's children will have 2 and 3. The second device that joins the coordinator will have an address of 4 and its children will be 5 and 6.   In more abstract terms, when the coordinator starts the network with the given parameters, then its address space will span from 0 to 6. It takes the '0' address for itself and then splits the remaining addresses among the routers that can join to it. In our case, the first router that joins gets the first half of the space and the second router that joins gets the second half. The following illustration demonstrates how the addresses are split up based on depth. The boxes in blue are the addresses that get taken by the device that owns the space.  ddress Allocation  Of course this allocation is only based on our simple test case with a maximum of 2 routers and a max depth of 2. Here's how the address is allocated if we change our parameters so we have a maximum of 3 routers and 3 children. We can't have more routers than total children since each router is considered a child. The depth is kept the same.  ddress Allocation - 3 Routers  In case you're wondering how I came up with the total address space, its actually quite simple. For the case with 2 routers, 2 children, and a network 2 deep, the total space is:  Total Space = 2^2 + 2^1 +2^0 = 7  For the case with 3 routers, 3 children, and 2 deep:  Total Space =3^2 + 3^1 + 3^0 = 13  If the number of children equals the number of routers, you can calculate the total number of devices from the following:  http://freaklabs.org/images/stories/2009-01-10%20Tree%20Routing/eqn1.jpg  And for the general case where the number of children is not equal to the number of routers (ie: end devices also exist) the max number of devices and hence the max value of the address space is:  http://freaklabs.org/images/stories/2009-01-10%20Tree%20Routing/eqn2.jpg  where Cm = Max End Devices  That's basically how the tree address allocation works for Zigbee. Now that we know how the addresses are doled out, it will be easier to see how they are used for routing.  **How Tree Routing Works**  If I didn't lose you on the address allocation scheme, then the routing should be a piece of cake. Now that we've established that each device has its own address space based on its depth, then we can determine how to route a frame based on its destination address. There are only two directions a frame can go in tree routing: up the tree or down the tree. If the destination address is within the device's address space, it should be routed down. Otherwise, it will be routed up.   Here's an example of a frame originating from device 2 with a destination address of 6. Each of the arrows are based on comparing the destination address with the device's address space (except the horizontal one which I just used to show the flow). If the destination address lies outside the device's address space, the frame gets routed up. Otherwise it gets routed down.  igbee Tree Routing  That's pretty much all there is to tree routing. It is heavily dependent on the addresses of each device and if you understand how addresses are allocated and the concept of each device having its own address space, then understanding how the frames get routed is pretty simple.  **The Problems with Tree Routing**  Tree routing is a clever and simple way of routing frames without a complex mechanism like AODV or some of the other routing algorithms out there. The big benefit is that it doesn't require tables so you can use tree routing in networks where the devices are very contrained on resources, ie: memory. However there is a fatal flaw in tree routing that renders it almost completely useless in a real world network.  The big problem is that this routing mechanism is fundamentally unreliable because the addresses are static. The position in the tree hierarchy of a device is set once that device joins the network and receives its address and afterwards, it's fairly inflexible to any changes to the network structure. So if a device joins the network and fails or moves out of range, then it will also take out all the devices underneath it in the hierarchy. It is possible to recover from this situation if all of the devices are able to rejoin the network with another parent successfully, however this can be a painful and error-prone process and potentially involve many devices.  Another fatal flaw is that there is no way to recover if the coordinator fails. It is the single point of failure for this routing method. In mesh routing, if the coordinator fails, things are still okay because from a mesh point of view, the coordinator is just another router in the routing tables.  However in tree routing, the coordinator is the vertex of the tree, where all frames must pass if the destination device is not located on the same branch as the source device. If the coordinator fails, then all the main branches become isolated and your network is pretty much useless. Currently, there's no place in the Zigbee spec (that I'm aware of) that describes how to recover from this failure.  From a reliability standpoint, tree routing is fairly dangerous. In fact, it was probably dangerous enough that the Zigbee Alliance decided to remove tree routing and addressing in the Zigbee Pro spec and I suspect that in the future, it may get taken out of Zigbee completely. In the FreakZ stack, tree routing is supported, but it's the frame forwarding method of last resort if all other methods fail.  Well, I hope this little blurb on tree routing was helpful. I don't really mean to knock on the algorithm since I think its pretty clever, but unless the Zigbee spec can address some of its reliability flaws, I would consider it dangerous to use. I guess what I'm implying in this article is that Zigbee shouldn't be used in beacon mode.  How do you like them apples... |

|  |  |
| --- | --- |
| Zigbee Mesh routing - Interactive Tutorial |  |

|  |
| --- |
| Written by Akiba |
| Tuesday, 08 April 2008 |
| Man. I spent the last three days learning how to use Flash so I could create this simple tutorial. You'd think that it would be easy to create a simple interactive thing like this, but it was actually extremely painful. You can't just draw the symbols and move things around. You actually need to program in a language that Adobe (MacroMedia) made called ActionScript. The programming part is no problem, but searching the massive documentation to find the right method to do what I wanted caused me to pound a large hole in the wall with my head. I still don't know why it requires a zillion languages just to do some stuff on the web.  Anyways, this is something that I've been wanting to do for a long time. Trying to explain Zigbee mesh routing using just words is very difficult, and has the tendency to cause people's eyes to glaze over. With an interactive simulation, you can do a play-by-play analysis of what's going on which hopefully will make it easier to see the flow of events that makes up the AODV routing algorithm. I originally had the idea when I was studying the routing algorithm myself. It's almost impossible to understand what's going on by reading the Zigbee spec. That's like trying to understand how to use Linux by reading the source code. You need to be a real masochist.  There weren't many good tutorials on the web for AODV mesh routing, either. So to save the people the same pain I had to go through in studying it, I created this little applet. To move forward one frame, just click anywhere on the graphic. To move back, press the space bar. Flash doesn't have a handler for the right-click button, because Apple users usually only have one mouse button. You can't tell who will be using the Flash file since it's accessed inside a web browser. You'll need to have Adobe Flash v9 installed (or higher depending on when you're reading this) in order to use it. Otherwise, it will probably just give you a bad graphic, no graphic, or it will cause your computer to explode.  I'm also planning to try and make other tutorials similar to this for tree routing, source routing, and possibly some other aspects of Zigbee. I think its much easier to learn visually than reading that monster document. Without further ado, here's the tutorial. Hope you like it. If there are any problems, leave a comment for me...especially if you find bugs.    Once you understand how the Zigbee routing works, then you can see that the terms that are being thrown around like "self-healing" are more a property of the routing algorithm. If a node goes down, then it won't participate in the route discovery process and so a different route will automatically be discovered. Okay, in my scenario, there were no redundant routers to the destination so that doesn't count.  Also, there are other subtleties associated with the algorithm used in Zigbee such as the use of "link cost" to determine the best route. This is the probability that the link may successfully transmit the frame and is a factor of interference, remaining battery life, and a gazillion other things. Its mainly subjective and depends on the company that does the implementation. There's no agreed upon way of calculating this so the spec just kind of leaves it as a ranking between 1 and 7 on how confident you think the frame will get transmitted. In my initial implementation, I'm just going to leave it as a static value. The path cost will then just reflect the number of hops from the source to the destination and the path with the least amount of hops will be chosen. Yes, it’s not very academic, but hey, I just want to get it to work. I'll let other people tinker with optimizing the routing depending on their needs.  Well, that just about does it for this post. Unfortunately, the research behind it took much longer than actually writing it, but at least future simulations will be much easier to make. Hope you enjoy. |