7

ZigBee

7.1 Development of the Standard

The 802.15.4 standard provides a physical and link layer technology optimized for low bitrate, low duty cycle applications. However, in practice sensor and control applications also need a mesh networking layer, and a standard syntax for application layer messages. In 2002, several companies decided to form the ZigBee alliance to build the missing standard layers that would be required to enable a multivendor mesh network on top of 802.15.4 radio links.

In 2008, the ZigBee alliance counted more than 200 members:

- *Promoter* members get early access to, contribute to and vote on the specifications of the alliance. They can veto decisions made by other participants in the alliance and get special marketing exposure in ZigBee events. New candidates for the promoter status must get co-opted by existing promoter members.
- *Participant* members have the same contribution and voting rights as promoters, but without veto rights.
- *Adopters* also get early access at the specifications, but can contribute only to the application profile working groups, and do not have voting rights.

The ZigBee alliance regularly organizes interop events, called ZigFests, and organize a developers conference twice a year. In order to ensure interoperability across vendors, the use of the ZigBee Compliant Platform (ZCP) certification and logo is reserved for products passing the ZigBee test suite, which includes interoperability tests with the "Golden units" (stacks from four reference implementations: Freescale, Texas Instruments, Ember, and Integration).

The deployment of many telecom standards either failed or was slowed down by multiple patent claims, many of which were not disclosed during the design phase of the

The Internet of Things: Key Applications and Protocols, First Edition.
Olivier Hersent, David Boswarthick and Omar Elloumi.
© 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd.

standard. While the ZigBee alliance can do nothing against potential patent claims coming from nonmembers, it did verify that no technology included in the standard was subject of a known patent. In addition, every new member of the ZigBee alliance must sign a disclosure statement regarding patents that could potentially apply to ZigBee technology.

There are several versions of ZigBee. The current versions of ZigBee are ZigBee 2006/2007 (stack profile 0x01, ZigBee 2007 adds optional frequency agility and fragmentation), and ZigBee Pro (stack profile 0x02) that adds support for more nodes and more hops through source routing (it does not support tree routing), multicasting, symmetric links and a high security level. There was a ZigBee 2004 version, which is now deprecated.

7.2 ZigBee Architecture

7.2.1 ZigBee and 802.15.4

ZigBee sits on top of 802.15.4 physical (PHY) and medium-access control (MAC) layers, which provide the functionality of the OSI physical and link layers.

So far ZigBee uses only the 2003 version of 802.15.4. All existing ZigBee commercial devices use the 2.4 GHz S-Band as the 2003 version of 802.15.4 does not allow sufficient bandwidth on other frequencies. The 2006 version adds improved data-transfer rates for 868 MHz and 900 MHz but is not yet part of the ZigBee specification.

802.15.4 offers 16 channels on the 2.4 GHz, numbered 11 to 26. ZigBee uses only the nonbeacon-enabled mode of 802.15.4, therefore all nodes use CSMA/CA to access the network, and there is no option to reserve bandwidth or to access the network deterministically. ZigBee restricts PAN IDs to the 0x0000 – 0x3FFF range, a subset of the 802.15.4 PAN ID range (0x0000-0xFFFE).

All unicast ZigBee commands request a hop by hop acknowledge (optional in 802.15.4), except for broadcast messages.

7.2.2 ZigBee Protocol Layers

The ZigBee network layer provides the functionality of the OSI network layer, adding the missing mesh routing protocol to 802.15.4. It also encapsulates the network formation primitives of the 802.15.4 MAC layer (network forming and joining).

The rest of the ZigBee protocol layers (Figure 7.1) do not follow the OSI model:

- The **Application Support Sublayer (APS)** layer has several functions:
  - Multiplexing/demultiplexing: it forwards the network layer messages to the appropriate application objects, according to their endpoint ID (each application is allocated an endpoint ID).
  - Binding: the APS layer maintains the local binding table, that is, records remote nodes and endpoints which have registered to receive messages from a local endpoint.
  - 64-bit IEEE to 16-bit ZigBee network node address mapping.

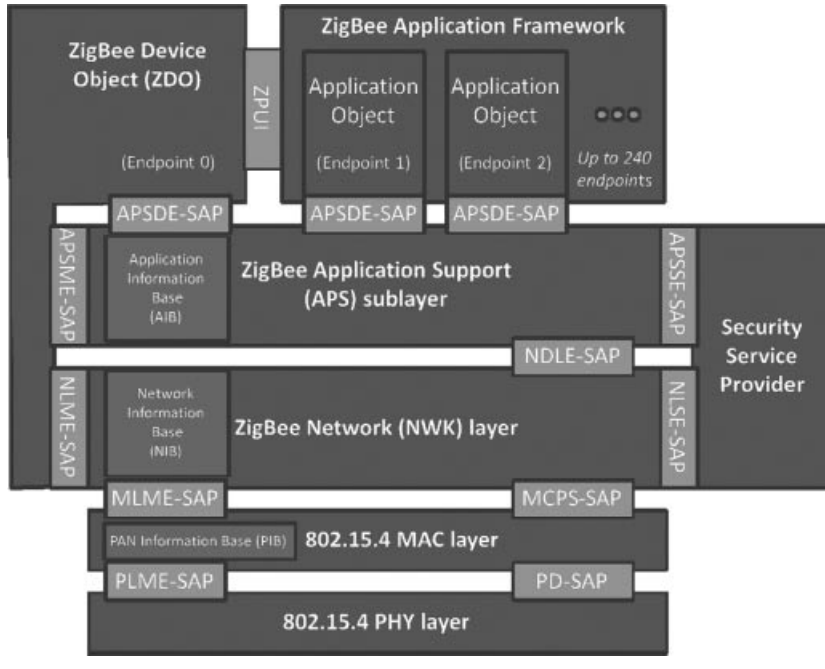

Figure 7.1 ZigBee architecture overview.

- Management of end to end acknowledgements. The application layer supports acknowledgements independently of the link layer acknowledgements of 802.14.4. The APS manages retries and duplicate filtering as required, simplifying application programming.
- Fragmentation.

  Also, as part of the application support sublayer management entity, or APSME:

- Group addressing: the APSME allows to configure the group membership tables of each endpoint ID, and forwards messages addressed to a group ID to the application objects with relevant endpoint IDs.
- Security: management of keys.

– The **ZigBee Device Object (ZDO)** layer is a specific application running on endpoint 0, designed to manage the state of the ZigBee node. The ZDO application implements the interfaces defined by the ZigBee device profile (ZDP, application profile ID 0x0000). These primitives encapsulate the 802.15.4 network formation primitives of the ZigBee network layer (node discovery, network joining), as well as additional primitives supporting the concept of binding (see Section 7.5.2.2).
– The **ZigBee Cluster Library (ZCL)** was a late addition to ZigBee, specified in a separate document. It consists in a library of interface specifications (cluster commands and attributes) that can be used in public and private application profiles. It is now

considered as one of the key assets of ZigBee: while the ongoing evolution of ZigBee towards a 6LoWPAN-based networking layer is likely to replace the original networking layers of ZigBee, the ZCL is likely to remain the "lingua franca" of application developers. One important addition of the ZCL is the group cluster, which provides the network interface for group formation and management.
- The **Application Framework** layer provides the API environment of ZigBee application developers, and is specific of each ZigBee stack. Each application is assigned an Endpoint ID.

The interfaces of ZigBee layers are called "service access points" (SAP), as in 802.15.4. One interface, the layer management entity ([layer name]-ME) is responsible for configuring internal data of the layer. Another interface, the data entity ([layer name]-DE), provides the data send/receive and other nonmanagement primitives.

7.2.3 ZigBee Node Types

The ZigBee node types listed below are not mutually exclusive. A given device could implement some application locally (e.g., a ZigBee power plug) acting as a ZigBee End Device, and also be a ZigBee router and even a ZigBee coordinator.

- **ZigBee End-Device (ZED):** this node type corresponds to the 802.15.4 reduced function device. It is a node with a low duty cycle (i.e. usually in a sleep state and not permanently listening), designed for battery operation. ZEDs must join a network through a router node, which is their parent.
- **ZigBee router (ZR):** this node type corresponds to the 802.15.4 full function device (FFD). ZigBee routers are permanently listening devices that act as packet routers, once they have joined an existing ZigBee network.
- **ZigBee Coordinator (ZC):** this node type corresponds to a 802.15.4 full function device (FFD) having a capability to form a network and become a 802.15.4 PAN coordinator. ZigBee coordinators can form a network, or join an existing network (in which case they become simple ZigBee routers). In nonbeacon-enabled 802.15.4 networks, coordinators are permanently listening devices that act as routers, and send beacons only when requested by a broadcast beacon request command.

The ZigBee coordinator also contains the trust center, which is responsible for admission of new nodes on the network and management of security keys (see Section 7.7).

7.3 Association

7.3.1 Forming a Network

When forming a network, a ZigBee coordinator first performs an active scan (it sends beacon requests) on all channels defined in its configuration files. It then selects the

channels with the fewer networks, and if there is a tie performs a passive scan to determine the quietest channel. It finally broadcasts a 802.15.4 beacon for the selected PAN ID on the selected radio channel, then remains silent (or repeats the beacon periodically, depending on the implementation). Depending on the configuration of the stack, the scan duration on each channel can range from 31 ms to several minutes, so the network-forming process can take significant time. If there are any ZigBee routers associated to the network, they will typically repeat the beacon with an offset in time relative to their parent's beacon (an extension of 802.15.4:2003).

The ZigBee specification allocates range 0x0000 to 0x3FFF for PAN IDs (a subset of the range defined by 802.15.4: 0x0000 to 0xFFFE). The PAN-ID should be unique for a given channel for networks not capable of dynamic channel change (ZigBee 2006), and unique on all channels if channel agility is enabled (ZigBee 2007, ZigBee PRO). A ZigBee coordinator beacon may also include an extended PAN ID (64 bit EPID), in addition of the 16-bit 802.15.4 PAN identifier, in order to facilitate vendor specific network selection for joining nodes. This EPID identifier is only used in the beacon frames and has no other uses, while the 16-bit 802.15.4 PAN identifier is always used for joining and addressing purposes.

7.3.2 Joining a Parent Node in a Network Using 802.15.4 Association

ZigBee devices that are not yet associated either capture by chance the beacon, or try to locate a network by broadcasting a 802.15.4 beacon request on each of the 16 radio channels (active scan, see Figure 7.2), unless the radio channel has been preconfigured or determined in the application profile. If a coordinator has formed a network on one of those channels, it responds to the beacon request by broadcasting a 802.15.4 beacon, which specifies the 16-bit PAN ID of the network, the address of the coordinator in short 16-bit format or extended 64-bit format, and optionally an extended PAN ID (EPID). Any ZigBee router that has already joined the network will also respond with a beacon if they hear the beacon request.

The ZigBee payload of the 802.15.4 beacon also contains the ZigBee stack profile supported by the network, a flag indicating whether the responding node has remaining capacity for routers or end devices joining as new children, and the device depth of the sending device, that is, its level in the parent/child tree rooted at the coordinator.

Once it has discovered the PAN ID of the network, its radio channel, and the address of a router or coordinator within radio reach, the new ZigBee node sends a standard 802.15.4 *association request* command to the address of the specific parent node it wants to join as a child node (0x01 profile nodes must join the node with the smallest device depth). The association request message uses the extended 64-bit address of the joining node as the source address. Devices may wish to join a specific PAN ID, or may use a special PAN ID value 0xFFFF to signal that they are willing to join any PAN ID.

The parent node acknowledges the command, and then if it accepts the association responds with a 802.15.4 association response command sent to the extended address of the device. The association response specifies the 16 bit short address that the device

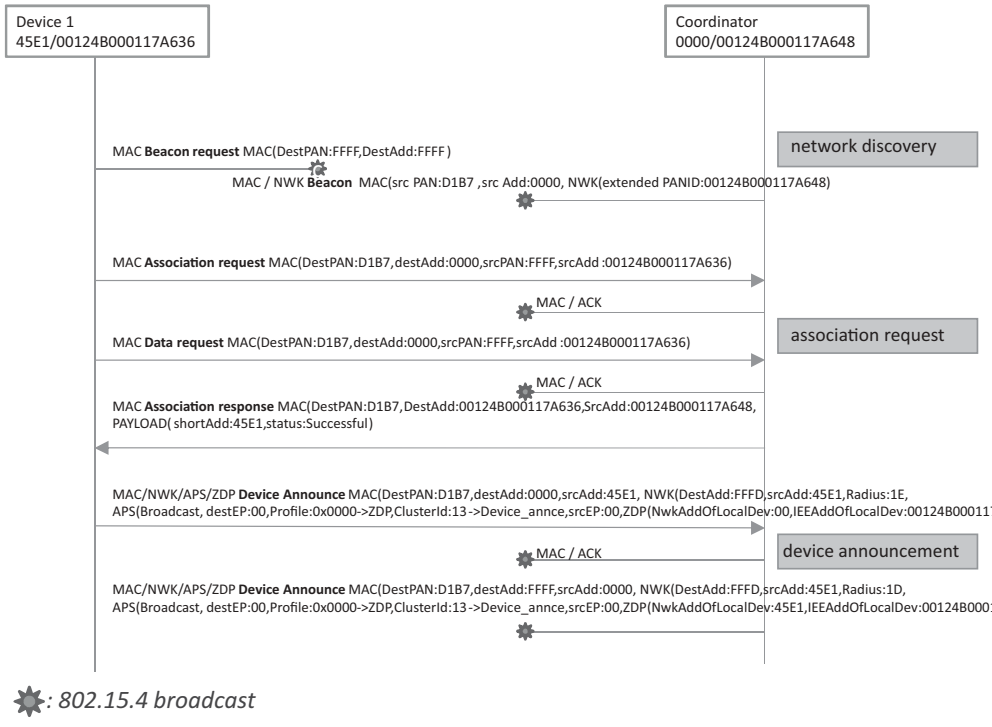

Figure 7.2 End device 1 joins the ZigBee network.

should use in the future (in order to save transmission time and therefore energy). The association response is acknowledged by the device. When the joining device is a sleeping node (RxOnIdle=false), the association response is not sent immediately, but stored until the sleeping nodes polls it using a "data request", as in the example of Figure 7.2.

Once associated, ZigBee devices usually send a data request command (now using the short 16-bit address assigned by the parent as source address) to its parent in order to receive any pending configuration data. After waiting for a response, battery-powered ZigBee end devices go to sleep until the next scheduled wake-up time or interrupt.

In the example of Figure 7.2, the joining device is within radio range of the coordinator. In a more general case, broadcast and unicast messages will be relayed by one or more ZigBee routers, and a new joining device may join the network from any location accessible through mesh networking.

ZigBee joining, at the lowest level of security, only uses the "permit joining" flag of the beacon: nodes can join a network only when this flag is set in the beacon response. At the application level, most implementations allow administrators to "permit joining" for a limited amount of time, after which the network will not accept further joins. If the device is allowed to join and the nwkSecurityLevel parameter is set to 0x00, then the node becomes a new child of the parent node with relationship type 0x01 (child), otherwise set

as type 0x05 (unauthenticated child). When security is enabled, interactions with the trust center (see Section 11.2.1) follow the unauthenticated joining process for key distribution.

In order to facilitate commissioning, nodes may implement the *commissioning ZDP cluster* (see Section 7.6) to preconfigure security material and other parameters, and reset the node. Some nodes may be set up to join any network with permit-join enabled, or may be preconfigured to join the well-known commissioning network with extended PAN ID 0x00f0c27710000000.

In theory, up to 31 100 nodes (9330 routers) can join a given network in stack profile 0x01, and over 64 000 nodes in stack profile 0x02.

7.3.3 Using NWK Rejoin

A device that loses connection to the network can attempt to rejoin using the ZigBee NWK layer rejoin command, which also triggers a beacon request. Since the NWK layer rejoin command use NWK layer security, the difference from a join based on 802.15.4 association is that no additional authentication step needs to be performed when security is enabled, and that nodes may rejoin any parent as long as it has available capacity, regardless of the status of the accept joining flag of the beacon. If it rejoins a different parent (e.g., because the original parent no longer responds), the node will be allocated a different short address, and must broadcast a device announce to the network in order to update bindings that may be configured in other nodes (see Figure 7.2).

After power cycles, most implementations do not immediately attempt an explicit rejoin in order to avoid network overloads, if they still have the address of their parent node and their own short address in nonvolatile memory. It is assumed that all nodes will restart in the same state as before the power cycle. An explicit rejoin is triggered only if the node fails to communicate with its parent. Such a procedure is often referred to as "silent rejoin". It is also the default procedure, in ZigBee Pro/2007, when the coordinator triggers a channel change (annex A).

7.4 The ZigBee Network Layer

The network layer is required for multihop routing of data packets in the mesh network, and is one of the key missing elements of 802.15.4. ZigBee uses the AODV public-domain mesh algorithm. The ZigBee network layer uses a specific data frame format, documented in Table 7.1, which is inserted at the beginning of the 802.15.4 payload.

7.4.1 Short-Address Allocation

ZigBee uses the 0x0000 – 0xFFF7 range for network node short addresses. The ZigBee coordinator uses short address 0x0000. The allocation of other network

addresses, under control of the ZigBee Coordinator, depends on the routing technology in use:

ZigBee supports two address allocation modes:

- In stack profile 0x01, the network address depends on the position of the node in the tree. The distributed address assignment mechanism uses CSkip, a tree-based network address partition scheme designed to provide every potential parent with a subblock of network addresses. In addition to the default meshed routing, a tree-based routing can be used as a back-up (routers use the address allocation to decide whether to forward the packet to a parent or to a child).
- In stack profile 0x02 (ZigBee 2007, ZigBee Pro), a stochastic address assignment mechanism is used and ZigBee provides address-conflict detection and resolution mechanisms.

7.4.2 Network Layer Frame Format

The network layer PDU format is illustrated in Table 7.1, and is transported as 802.15.4 payload (see Chapter 1).

Table 7.1 The ZigBee network layer frame format

Field name	Size (octets)	Field details
Frame Control	2	----------XX : Frame type (00 : network data) ----------0010-- : Protocol version (always 0x02 for ZigBee 2006/2007/Pro) --------XX------ : Route discovery (0x01:enable) -------X-------- : Multicast (0 : unicast) ------X--------- : Security (0 : disabled) -----X---------- : Source route (0 : not present) ----X----------- : Destination IEEE address (0 : not specified) ---X------------ : Source IEEE address (0 : not specified) 000------------- : Reserved
Dest. Address	2 or 8	0xffff broadcast to all nodes including sleeping devices 0xfffd broadcast to all awake devices (RxOnIdle = True) 0xfffc broadcast only to routers, not to sleeping devices
Source address	2 or 8	
Radius	1	Maximum number of hops allowed for this packet
Sequence number	1	Rolling counter
Payload	Variable	APS data, or network layer commands

7.4.3 Packet Forwarding

At the network layer, ZigBee packets can be:

- *Unicast*: the message is sent to the 16-bit address of the destination node
- *Broadcast*: if broadcast address 0xFFFF is used, the message is sent to all network nodes. If broadcast address 0xFFFD is used, the message is sent to all nonsleeping nodes. If broadcast address 0xFFFC is used, the message is broadcast to routers only (including the ZigBee coordinator). A radius parameter adjusts the number of hops that each broadcast message may travel. The number of simultaneous broadcasts in a ZigBee network is limited by the size of the broadcast transaction table (BTT), which requires an entry for each broadcast in progress. The minimal size of the BTT is specified in ZigBee application profiles, for example, 9 for HA.
- *Multicast* that is, sent to a 16-bit group ID.

ZigBee unicast packets are always acknowledged hop by hop (this is optional in 802.15.4). Broadcast packets are not acknowledged and are usually retransmitted several times by the ZigBee stacks of the originating node and by ZigBee routers on the path. Note that broadcast messages and messages sent to group IDs are not always broadcast at the link layer level: since sleeping nodes do not receive such messages, after wake up they send a data request to their parent node, and the queued messages are sent as unicast messages specifically to the sleeping node (in which case they are acknowledged at the link-layer level). In most implementations, the messages are queued only 7 to 10 seconds in the parent node (ZigBee specifies 7 seconds) so sleeping nodes should wake up at least once every 6 seconds. However in practice several vendors manufacture sleeping nodes with wake up periods of up to 5 minutes . . . in this case applications need to be prepared to resend commands every 6 seconds until the target node wakes up.

Typically, a ZigBee node forwards a packet in about 10 ms. The propagation of data packets through the meshed network is limited by the initial value of Radius, a hop counter decremented at each routing node.

Delivery of packets to sleeping nodes uses the IEEE “Data request” packet. Parent nodes buffer received packets for their sleeping children. When it wakes up, the sleeping child sends a IEEE “Data request” packet to its parent. If it has data pending, the parent sets a specific “more data” flag in the ACK response, then the pending data.

7.4.4 Routing Support Primitives

The network layer provides a number of command frames listed in Table 7.2.

The route request command enables a node to discover a route to the desired destination, and causes routers to update their routing tables. At the MAC level, the route request command is sent to the broadcast address (0xffff) and the current destination PAN ID.

The response, if any, is a route reply command that causes routers on the path to update their routing tables.

Table 7.2 ZigBee routing layer primitives

Command Frame Identifier	Command Name Reference
0x01	Route request
0x02	Route reply
0x03	Network Status
0x04	Leave
0x05	Route record
0x06	Rejoin request
0x07	Rejoin response
0x08	Link status
0x09	Network report
0x0a	Network update

7.4.5 Routing Algorithms

7.4.5.1 Broadcast, Groupcast, Multicast

At the 802.15.4 level, messages sent to multiple destinations are always broadcast. At the network layer, however, ZigBee offers more possibilities, depending on the destination address:

- 0xffff broadcast to all nodes including sleeping devices;
- 0xfffd broadcast to all awake devices (RxOnIdle = True);
- 0xfffc broadcast only to routers, not to end devices.

In order to avoid 802.15.4 collisions, broadcast packet are relayed after a random delay of about 100 ms and therefore propagate ten times more slowly than unicast messages. The radius parameter is decremented at each hop, so the broadcast propagation can be controlled with the initial radius value (see Table 7.1).

ZigBee also uses a broadcast transaction table (BTT) in order to avoid any looping of broadcast messages: each broadcast packet is uniquely identified by its source address and network sequence number. When relaying a broadcast message, routers keep a copy of this unique identifier for 9 s (broadcast timeout), and will drop any looped packet. If the BTT is full, all broadcast messages are dropped. Routers that do not hear all neighbor routers retransmit a broadcast message may retransmit the broadcast message, implementing a form of implicit acknowledge mechanism.

Groupcast and multicast are implemented by the APS layer:

- APS messages sent to group addresses are filtered by the APS layer of the receiving node, so that only endpoints (and all of them) of member nodes will receive the message. However, all nodes receive the message (destination set to 0xffff at the network layer)

– In ZigBee Pro, the radius is not decremented when the message is forwarded by a group member. This makes it possible to restrict the 802.15.4 broadcast propagation to group members only, allowing some slack (apsNonmemberRadius) in order to cope with disconnected groups. ZigBee Pro calls this “multicast”.

7.4.5.2 Neighbor Routing

This mode is not formally documented in ZigBee, but most vendors use it. If a router R already knows that the destination of a packet is a neighbor router or a child device of R, it can send the packet directly to this node. ZigBee end devices, however, must always route outgoing packets to their parent.

7.4.5.3 Meshed Routing

This is the default routing model of ZigBee. It implements the advanced *ad-hoc* on-demand distance vectoring (AODV) algorithm.

The principle of AODV is illustrated on Figure 7.3.

Node A needs to set up a route to node D. It broadcasts a route request (see Table 7.3) to network address 0xfffc (routers only), which propagates through the network. Each ZigBee router that receives that message forwards it to its neighbors, adding their local estimation of quality of the link over which they received the route request to the path cost parameter of the route request. Note that the route we are discovering is in the A to D direction, while the path costs actually used are in the D to A direction. This is because the sender of the route request, which is broadcasting the message, cannot transmit different values of the link cost, therefore the receiving node needs to update the path cost. *ZigBee mesh routing assumes symmetrical link quality*.

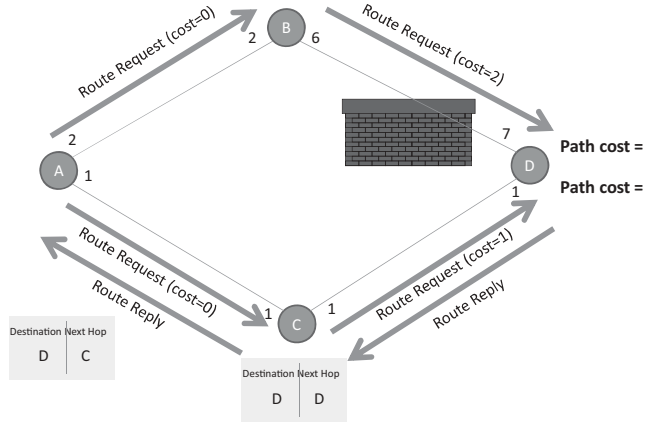

Figure 7.3 ZigBee mesh route discovery.

Table 7.3 Route request parameters

Command options	Route request identifier	Destination address	Path cost	Destination IEEE address
(1 octet)	(1 octet)	(2 octets)	(1 octet)	(0 or 8 octets)
xxx00xxx: not a many-to-one route request **xxx01xxx**: many-to-one route request and the sender supports a route record table **xxx10xxx**: many-to-one route request and the sender does not support a route record table **xxxxx1xx**: the command frame indicates the destination IEEE address, otherwise set to 0 **xxxxxx1x**: route request for a multicast group, and the destination address field contains the group ID, otherwise set to 0	Sequence number	Intended destination	Accumulator for path length as the command is propagated	Only if bit 5 of the command option is set to 1

Node D will wait for a while until it believes it has received all broadcast route requests, then computes the lowest path cost and responds by a unicast route reply along the best route that was just discovered (the parameters of the response are listed in Table 7.4). Router C creates a routing table entry for D, recording the next hop to reach D (in this simple case, D itself), and node A also creates a routing table entry for D, recording C as the next hop router.

Table 7.4 Route response parameters

Command options	Route request identifier	Originator address	Responder address	Path cost	Originator IEEE address	Destination IEEE address
(1 octet)	(1 octet)	(2 octets)	(2 octets)	(1 octet)	(0 or 8 octets) Address of the originator of the route request.	(0 or 8 octets) Address of the responder.

Node A will continue to use this route as long as it works, or until the applications requests calculation of a new route.

In ZigBee 2006 and 2007, routes are unidirectional: D will need to discover a route if it needs to send a packet to A. In ZigBee Pro, D would also store the route to A, assuming symmetry. ZigBee Pro routers "ping" each other every 15 s to make sure links are indeed bidirectional, and eliminate one-way links for both directions.

7.4.5.4 Tree-Based Routing

Tree-based routing is a back-up routing mechanism (when no route exists or can be discovered) that can be used only in ZigBee 2006/2007 networks which use Cskip-based address allocation. Tree-based routing simply uses the fact that routers know the blocks of addresses allocated to their router children, and to their children ZigBee end devices: if the destination address is one of those, the router forwards the packet to the appropriate child, otherwise it propagates the packet to its parent.

Because of the limited address space of 802.15.4 (64 K addresses), which limits the size of CSkip address blocks, tree-based routing is limited to 5 parent/child relationships, each node having a maximum of 20 children and 6 routers. This limits the number of nodes to 31 101 in ZigBee 2006/2007.

7.4.5.5 Source Routing

ZigBee mesh networking is limited by the size of the routing table of routers. It works and scales well in a peer to peer environment, but in environments where one node is a preferred communication source that frequently communicates to all other nodes, then this node and adjacent routers would need to store a routing table with as many entries as nodes. This situation happens with data concentrators, used for metering applications.

ZigBee Pro solves this problem by introducing source routing:

- The concentrator broadcasts a many to one route request (up to five hops), which enables routers along the path to record the shortest path to the concentrator.
- When a node first sends a packet to the concentrator, it first sends a route record message towards the concentrator, so that the concentrator will have a chance to learn the optimal path back towards the node (assuming symmetric links).
- The concentrator can now reach any node using source-routed messages (up to 5 intermediary routers can be specified). It still needs a lot of memory, but all routers in the ZigBee network can now work with very small routing tables.

7.5 The ZigBee APS Layer

The APS layer is responsible for management and support of local applications. It defines all the concepts that make it possible to develop and interconnect ZigBee applications: endpoints, groups, bindings, and so on.

7.5.1 Endpoints, Descriptors

A given ZigBee device may implement multiple applications at the same 802.15.4 address. Clearly some multiplexing mechanism is required to identify the source and destination application of a message. This multiplexing identifier is called *endpoint* in the ZigBee specification. Think of it as the equivalent of a port number in a TCP/IP network.

Each endpoint is further characterized by a simple descriptor. The simple descriptor contains the endpoint number, application profile ID, application device ID (a 16-bit number referring to a device definition, for example, a HA thermostat, used only for informative purposes since it contains no technical data), an application version ID, the list of input clusters and the list of output clusters.

The descriptor of an endpoint can be retrieved from any other node by using the ZDP simple descriptor request command (Simple_Desc_req, see also Section 7.8.1).

In addition to the simple descriptor, specific to each endpoint, ZigBee devices have additional descriptors applying to the whole node:

- A node type descriptor: capabilities of the node;
- A node power descriptor: node power characteristics;
- A complex descriptor (optional): Further information about the device descriptions, described as pairs of compressed XML tag and related field data;
- A user descriptor: User-definable descriptor.

Profiles can be discovered by using the ZigBee device profile request primitive, addressed to endpoint 0 (ZDO) of the device.

7.5.2 The APS Frame

APS data frames can be sent unicast (with or without application level end to end acknowledgment, in addition to the MAC level hop per hop acknowledgment), groupcast, multicast (ZigBee 2007 and ZigBee Pro), or broadcast. Groupcasts and broadcasts are both supported by network-level broadcasts, and are not acknowledged.

At the application level, ZigBee allows application developers to use 64-bit or 16-bit addresses, group addresses or indirect addressing in order to identify the destination node, for instance in the *APSDE-DATA.request*. In all cases, the ZigBee stack resolves that address to a 16-bit node address or to a group address before transmission.

This resolution mechanism uses the APS address map. This cache stores the mapping of 64-bit IEEE addresses to 16-bit ZigBee short network addresses. It is used, for instance, to resolve binding requests (which specify only a 64-bit IEEE address) to a 16-bit address. The maintenance of this table is performed by listening to broadcast device announce commands (e.g., when a device changes location and its 16-bit address changes, see Figure 7.2 for an example).

If a node does not have a cached route to the destination, it performs a route discovery using ZDP commands `IEEE address request` and `NWK address requests`. When a frame that required an end to end acknowledgment has not been acked after 3 retries (typically a retry every 1.5 s, this delay is adjustable in most stacks), another route discovery may be performed.

7.5.2.1 Groups

A group identifier (in the range 0x0000 to 0xFFFF) is an address that can be used at APS layer level to send a message to multiple ZigBee applications residing on other nodes (see Section 7.4.5.1). Any ZigBee node can belong to up to 16 groups. An application residing on a ZigBee node on endpoint E adds itself to a group G by calling the local AddGroupRequest APS primitive: this function adds endpoint E to the list of local members of group G.

Messages addressed to a group are broadcast at the ZigBee network layer (the ZigBee network frame destination address is 0xffff)[1]: ZigBee routers will forward a copy of the packet to any neighbor. Group messages are therefore received and processed by all nodes in the PAN network at the MAC layer, but the APS layer forwards the message only to the endpoints (individual applications residing on the node) that have registered to be members of the group ID.

The ZigBee HA profile recommends group addressing each time a message needs to be sent to more than 4 nodes.

7.5.2.2 Indirect Addressing, Binding

Bindings are one of the publish/subscribe models implemented in the ZigBee specification (together with attribute reporting, see Section 7.8).

Cluster C (see Section 7.8) on source endpoint E1 is bound to destination endpoint E2 (typically hosted by a different node) if it sends events related to its output cluster ID(s) to the corresponding input cluster ID(s) of E2. E1 can be bound to multiple target endpoints. Each binding is unidirectional and independent, if E1 is bound to E2, E2 may or may not be bound to E1 (it is a totally different binding).

The binding table can be managed locally through an API (APSME-BIND.request) or remotely via ZDP commands: The ZDP end-Device-Bind request is sent to endpoint 0 of the target node and specifies:

- The target endpoint of the binding;
- The source 64-bit IEEE address (the 16-bit ZigBee network address is resolved by the APS network address map);

[1] At the 802.15.4 level the ZigBee group messages might be unicast (if the sending node only has one neighbor) or broadcast.

- The source endpoint;
- The list of input clusters and output clusters of the source endpoint.

The local binding table lists, for each binding:

- The local source endpoint;
- The application layer destination address that can be a 802.15.4 address (64-bit format), or a group ID (16-bit);
- The destination endpoint if the destination is not a group address;
- A cluster ID.

A typical use case is that a device looks for another node in the network with capabilities corresponding to a match descriptor (supported application profile, cluster ID and direction, e.g., a lamp supporting the on/off cluster as an input). It then binds to that node using the End-Device-Bind command. In order to facilitate this configuration operation, bindings may be specified for groups. ZigBee devices that can initiate or process events have a button that places them in "identify mode" for about 10 s. Command AddGroupIfIdentifying can be broadcast and will automatically place the nodes in "Identify" mode in the group.

At the application level, the binding table can be used through the indirect addressing mode, for example, in the *APSDE-DATA.request* primitive. When indirect addressing is used, the destination address (node address or group address) is resolved using the local binding table, based on the endpoint ID of the sending application. Indirect addressing is very flexible as it allows external nodes to configure the routing of messages across ZigBee applications residing on different nodes (e.g., instruct a switch to send its on/off events to a ZigBee-controlled relay).

7.5.2.3 APS Frame Format

The APS frame format is outlined in Table 7.5.

The format of the application level payload depends on the value of the application profile identifier and the cluster identifier:

- Application Profile ID 0x0000: the payload format is defined by the ZigBee device profile (ZDP).
- Application Profile IDs 0x0000 to 0x7FFF are reserved for public application profiles, the payload format is defined by the ZigBee cluster library (ZCL)
- Application Profile IDs 0xBF00 to 0xFFFF are reserved for manufacturer specific profiles (MSP). The payload format is defined by the manufacturer but may also use the ZCL.

Table 7.5 The APS frame format

Field name	Bytes	Field details
Frame Control	1	------XX : Frame type (00 : APS data) ----XX-- : Delivery mode (00 : unicast; 11 : group addressing) ---X---- : Indirect address mode (0 : ignored) --X----- : Security (0 : none) -X------ : Ack (0 : not required) 0------- : Reserved
Dest. Endpoint	1	16-bit destination address or group ID
Cluster identifier	2	0x0006: On/Off
Application Profile identifier	2	0x0104: HA
Source endpoint	1	
Counter	1	APS level counter
AF payload	Variable up to 80 bytes	APS service data unit (ASDU): a ZCL frame, ZDP frame or application-specific payload.

The ZDP and ZCL are described in the following Sections 7.6 and 7.8.

7.6 The ZigBee Device Object (ZDO) and the ZigBee Device Profile (ZDP)

The **ZigBee Device Object (ZDO)** layer is a specific application running on endpoint 0, designed to manage the state of the ZigBee node. The ZDO application implements the interfaces defined by the ZigBee device profile (ZDP, application profile ID 0x0000).

The clusters defined within the ZDP are similar to those defined in application-specific profiles, but unlike them, the clusters within the ZigBee device profile define capabilities supported in all ZigBee devices. All ZDP client-side transmission of cluster primitives is optional.

The ZDO implements a number of configuration attributes (e.g., the various node descriptors), as well as a number of local APIs and network primitives.

7.6.1 ZDP Device and Service Discovery Services (Mandatory)

These primitives support the discovery of nodes based on some of their characteristics. Since sleeping devices are not capable of receiving such requests, a cache mechanism is provided. The discovery cache is a database of nodes that registered to this cache after a find node cache request, and stores cached descriptor data (stored using Node_Desc_store_req).

ZDP primitives with mandatory server-side processing:

- NWK_addr_req (NWK_addr_rsp): finding a network address from a given IEEE address.
- IEEE_addr_req (IEEE_addr_rsp): finding an IEEE address from a given NWK address.
- Node_Desc_req, Power_Desc_req, Simple_Desc_req (and corresponding responses): finding the node/power/simple descriptor of a device from a given NWK address.
- Active_EP_req (Active_EP_rsp): acquire a list of endpoints on a remote device with simple descriptors.
- Match_Desc_req (Match_Desc_rsp): finding a list of devices with matching profile ids and cluster IDs.
- Device_annce.

ZDP primitives with optional server-side processing:

- Complex_Desc_req, User_Desc_req, Discovery_Cache_req, User_Desc_set, System_Server_Discover_req, Discovery_store_req, Node_Desc_store_req, Power_Desc_store_req, Active_EP_store_req, Simple_Desc_store_req, Remove_node_cache_req, Find_node_cache_req, Extended_Simple_Desc_req, Extended_Active_EP_req ... and corresponding responses.

7.6.2 ZDP Network Management Services (Mandatory)

These services implement the network features corresponding to the node type (coordinator, router or end device), for example, managing network scan procedures, interference detection, and so on. It provides the related interfaces for local applications. Remote nodes can also send a remote management command to permit or disallow joining on particular routers or to generally allow or disallow joining via the trust center.

Only the Mgmt_Permit_Joining_req/rsp server-side processing is mandatory, all other primitives are optional:

- Mgmt_NWK_Disc_req/ Mgmt_NWK_Disc_rsp (control network scanning).
- Mgmt_Lqi_req / Mgmt_Lqi_rsp (getting the neighbor list from a remote device).
- Mgmt_Rtg_req./ Mgmt_Rtg_rsp (getting the routing table from a neighbor device).
- Mgmt_Bind_req. Mgmt_Bind_rsp (getting the binding table from a neighbor device).
- Mgmt_Leave_req / Mgmt_Leave_rsp (request a remote device to leave the network).
- Mgmt_Direct_Join_req / Mgmt_Direct_Join_rsp (requesting that a remote device permit a device designated by DeviceAddress to join the network directly).
- Mgmt_Cache_req / Mgmt_Cache_rsp (allows to retrieve a list of ZigBee end devices registered with a primary discovery cache device).
- Mgmt_NWK_Update_req / Mgmt_NWK_Update_rsp (allows communication of updates to the network configuration parameters).

7.6.3 ZDP Binding Management Services (Optional)

These primitives enable binding management and maintenance of the bindings table (e.g., processes device replacement notifications). The concept of binding and indirect addressing is discussed in Section 7.5.2.2.

- End_Device_Bind_req / End_Device_Bind_res;
- Bind_req / Bind_res;
- Unbind_req / Unbind_res;
- Bind_Register_req / Bind_Register_res;
- Replace_Device_req/ Replace_Device_res;
- Store_Bkup_Bind_Entry_req/ Store_Bkup_Bind_Entry_res;
- Remove_Bkup_Bind_Entry_req/ Remove_Bkup_Bind_Entry_res;
- Backup_Bind_Table_req/ Backup_Bind_Table_res;
- Recover_Bind_Table_req / Recover_Bind_Table_res;
- Backup_Source_Bind_req / Backup_Source_Bind_res;
- Recover_Source_Bind_req / Recover_Source_Bind_res.

7.6.4 Group Management

Although one would expect group management over the network to be part of the core ZDP primitives, these were added later as part of the ZCL groups cluster (cluster ID 0x0004). See Section 7.8.

7.7 ZigBee Security

7.7.1 ZigBee and 802.15.4 Security

ZigBee networks can choose whether to enable security or not. Devices compliant with a public application profile must conform to their profile security settings.

ZigBee offers security services at two levels:

- Network (NWK) level security;
- Application (APS) level security.

None of these security services uses the MAC-level security defined by 802.15.4 (the 802.15.4 frame control field security bit is set to 0), which would encrypt the ZigBee NWK header that is required by ZigBee routing. However, ZigBee simply transposes the exact same mechanisms to the ZigBee network and application layer, and uses the

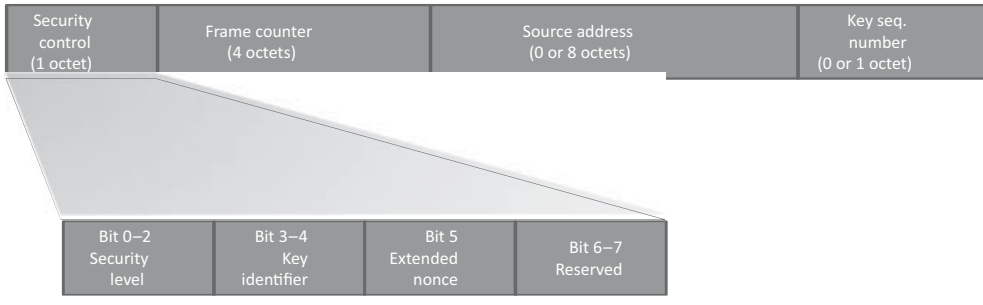

Figure 7.4 Auxiliary Security header format.

same encryption and hash algorithm (AES-CCM*), so encryption acceleration modules of 802.15.4 chips can still be used.

The ZigBee device object (ZDO) manages the security policies and configuration of a device.

7.7.1.1 NWK Level Security

ZigBee provides optional integrity protection and encryption, as illustrated in Figure 7.6. When network-level integrity protection or encryption is used, the security bit in the NWK control field is set to 1, indicating the presence of a security auxiliary frame header, and a message-integrity protection code (MIC).

The NWK security auxiliary header is composed of four subfields, illustrated in Figure 7.4.

In the security control subfield, the security level is set to the value of the MIB nwk security-level parameter, by default 0x05. This value specifies whether the frame is only integrity protected but not encrypted, or that it should also be encrypted, in which case the entire network payload is encrypted. Network security is applied as configured in the device network information base (NIB, nwkSecureAllFrames=TRUE to secure all frames) by default, but the application may override this setting frame by frame by specifying the SecurityEnable parameter of the NLDE-data.Request primitive.

The services provided by each security level are listed in Figure 7.5.

The MIC, as well as encryption, are computed using AES and CCS* (see Section 1.1), using the main or alternate network key.

7.7.1.2 Application Layer Security

The APS layer provides a number of security primitives that can be used by application developers:

- APSME-ESTABLISH-KEY to establish a link key with another ZigBee device using the SKKE protocol.
- APSME-TRANSPORT-KEY to transport security material from one device to another.
- APSME-UPDATE-DEVICE to notify the trust center when a device joins or leaves the network.
- APSME-REMOVE-DEVICE to instruct a router to remove a child from the network (used by the trust center).
- APSME-REQUEST-KEY to ask the trust center an application master key or the active network key.
- APSME-SWITCH-KEY used by the trust center to tell a device to switch to a new network key.
- APSME-AUTHENTICATE used by two devices to authenticate each other.

Security control	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
encryption	No				Yes			
MIC bit size	No MIC	32	64	128	No MIC	32	64	128

Figure 7.5 AUX header security control field values. The key identifier is set of 0x01 for the active network key (0x00 for link keys used by the APS layer).

In addition, the application-layer security provides its own optional integrity and encryption services, based on the network key or on a link-specific key (associated to the destination of the packet), under control of the application (TxOptions parameter).

Just like NWK security, the APS layer security will add an auxiliary security header (AUX header), and an integrity code, as illustrated on Figure 7.6. The difference is that the integrity protection scheme protects the APS header, auxiliary header and APS payload, and when encryption is used, only the APS payload is encrypted.

7.7.2 Key Types

- **Master Keys**
  - Application master keys are distributed by the trust center (via unsecured key transport) and used to set up link keys between two devices, and for mutual authentication of devices.
  - Trust center master keys are used to derive a link key for communication with the trust center.

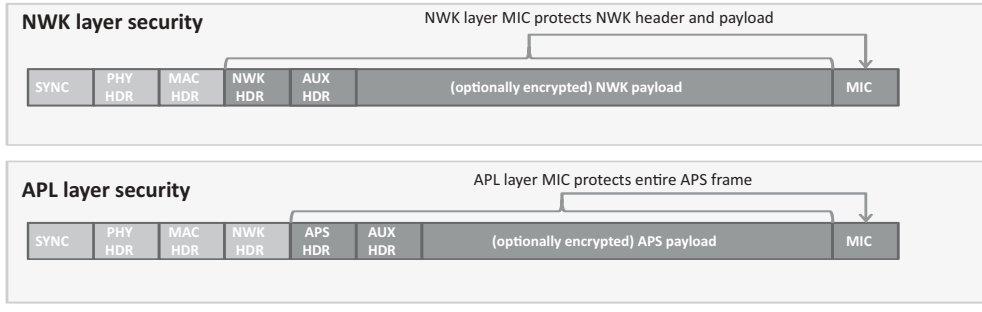

Figure 7.6 ZigBee NWK and APS security services.

- **Network keys** are used for network management. They are preconfigured or configured by the trust center, using unsecured key transport when standard security is used or using the trust center link key in high-security mode.
  - Standard network keys are used in the standard security mode, and can be used to secure general application layer commands.
  - High-security network keys are used in the high-security mode, they are not used for communication between secure devices, which use link-specific keys instead.
- **Link keys** can either be negotiated using SKKE, or configured by the trust center under request of a device. Service specific keys are derived by hashing of the link key:
  - The **key-load** key is used to protect transported master and link keys.
  - The **key-transport** key is used to protect transported network keys.

A ZigBee device stores a (master key/link key) key pair for each device with which they may use link-key-based communication. The device is identified by its 64-bit IEEE address.

7.7.3 The Trust Center

Key distribution is not addressed by 802.15.4. For security purposes, ZigBee defines the role of "trust center", which is responsible for key distribution and joining policy.

In *high-security mode*, the trust center maintains a list of devices, master keys, link keys and network keys. A device can be preloaded with the trust center address and initial master key, or the master key can be sent via an unsecured key transport primitive. The trust center, by default, is the ZigBee coordinator, but the coordinator can designate another device, or the trust center can be preconfigured in devices.

In *low-security mode*, a device communicates with its trust center using the current network key, which can be preconfigured (as in the commercial building automation profile) or configured via an unsecured key transport primitive during the joining process (as in the home automation profile).

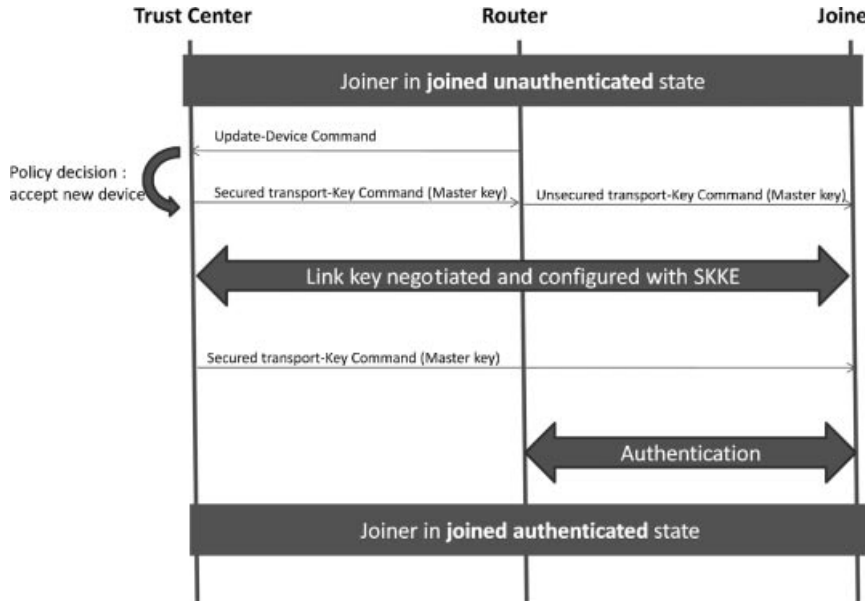

Figure 7.7 Security bootstrapping procedure in high-security mode.

The trust center is notified by ZigBee routers of joining devices by means of an Update-Device command (see Figure 7.7). They can reject or accept the new joiner. In the latter case the trust center communicates the joiner master key to the router, which relays it to the joiner. This enables the trust center to establish a link key to the joiner, and to finally securely communicate the NWK key to the joiner using a secure Transport-Key command (key-transport key, derived from the link key).

Devices can also request the trust center to compute a link key pair for communication with another device: the trust center will communicate the link key to each device using Transport-Key commands.

7.7.3.1 SKKE

Secure ZigBee devices are configured with a link specific master key for each device they may need to communicate securely with, or need to authenticate. A link key, distinct from the master key, can be negotiated between ZigBee endpoints by using a symmetric key establishment (SKKE) scheme using procedures defined in ANSI X9.63-2001.

7.7.3.2 Entity Authentication

ZigBee provides a APSME-AUTHENTICATE.request primitive is used for initiating entity authentication or responding to an entity authentication initiated by another device. The request includes a random challenge and the response is a hash based on the shared master key for the device pair and frame sequence numbers.

7.7.4 The ZDO Permissions Table

The ZDO optionally comprises a permissions configuration table. This table lists a number of tasks categories (e.g., *ApplicationSettings* for authorization to configure bindings, groups, and other application configuration commands) that can be requested from an external entity to the ZDO. For each task category, the permissions configuration table lists the addresses of devices authorized to perform these tasks (or specifies that any device is authorized), and specifies whether the related commands need to be secured with a link-specific key.

7.8 The ZigBee Cluster Library (ZCL)

The cluster library provides support for communication between applications, including over the air configuration of group tables, reading and setting attribute values, subscribing to certain attribute-value changes, and so on.

7.8.1 Cluster

A cluster is a set of related commands and attributes. Each cluster is defined by a ZigBee application profile and identified by a cluster ID (0x0000 to 0xFFFF). Commands are identified by a command ID (0x00 to 0xFF), and attributes by an attribute ID (0x0000 to 0xFFFF).

Clusters are composed of a server side and a client side. Each application residing on a ZigBee device lists the clusters that it supports as a server in its simple descriptor input cluster list, and the clusters that it supports as a client in the simple descriptor output cluster list.

An application that implements the server side typically exposes a set of attributes, and must be able to receive and process the primitives defined by the cluster to read or manipulate these attributes. It may also support attribute-reporting commands, which are sent to requesting devices that implement the client side of the cluster.

Vice versa an application that implements the client side of the cluster may use any of the commands supported by the server side, and must support receiving attribute reporting commands, if it requests such notifications.

Each cluster is defined only within a given application profile. However, in order to avoid any confusion and to redefine common clusters multiple times, the clusters defined by the ZigBee cluster library (ZCL) use the same cluster IDs for each ZigBee public application profile. Table 7.6 lists some of the clusters defined by the ZCL, which therefore have the same meaning for all public application profiles defined by ZigBee.

Clusters are directional: for each endpoint, the endpoint simple descriptor contains the list of input clusters (commands that it accepts and properties that can be written to the endpoint), and the list of output clusters (commands that it may send and properties that may be read).

Table 7.6 Some clusters defined by the ZCL

Cluster ID	Cluster Name
0x0000	Basic cluster
0x0001	Power configuration cluster
0x0002	Temperature configuration cluster
0x0003	Identify cluster
0x0004	Groups cluster
0x0005	Scenes cluster
0x0006	OnOff cluster
0x0007	OnOff configuration cluster
0x0008	Level control cluster
0x000a	Time cluster
0x000b	Location cluster

Each ZigBee public application profile defines a list of devices, each implementing a standard set of features defined by a list of input and output clusters. Within a given public-application profile, manufacturers can extend that list by using the "manufacturer specific extension" field of the ZCL frame (see Section 7.8.4). An endpoint declares the clusters it supports in its simple descriptor, which contains the endpoint number, application profile ID, application device ID, and application version ID (see Section 7.9), the list of input and output clusters.

7.8.2 Attributes

Attributes are identified by a 16-bit number. The ZCL library provides commands to:

- read or set attributes;
- require reporting on an attribute, either periodic or based on a change of value.

7.8.3 Commands

Commands are identified by an 8-bit number. The first 4 bits of cluster-specific commands (defined only within the scope of a given cluster) are set to zero. For instance, command 0x00 of the OnOff cluster (0x0006) means "Off".

Other identifiers are reserved for cross-cluster commands.

7.8.4 ZCL Frame

The ZigBee cluster library frame (Table 7.7) carries application-specific commands.

The general commands (frame type = 00) are listed in Table 7.8.

Table 7.7 ZCL frame format

Field	Bytes	Field details
Frame Control	1	------XX : Frame type (00 : cluster for entire app. Profile, 01 : cluster specific command, other values reserved) -----0-- : Manufacturer specific (0 : defined by ZCL, 1 manufacturer specific) ----X--- : Direction (0 : client to server) 000X---- : Default response (0 enabled, 1 disabled) 000----- : Reserved
Manufacturer code	0/2	Present only if the frame is manufacturer specific (Frame control bit 6)
Transaction sequence number	1	Matching of request command frames and response command frames
Command identifier	1	e.g., 0x42: toggle

Table 7.8 ZCL general commands

Command identifier	Command	Usage
0x00	Read attributes	Read a list of attributes, whose 16-bit identifiers are passed as payload
0x01	Read attributes response	Includes a list of attribute records as payload. Each attribute record is as follows: Attribute identifier (16 bit) | Status(Success or error: 8 bit) | Attribute data type (0/8 bit) | Attribute value (variable). Array types are encoded as element type | number of elements | list of elements.
0x02	Write attributes	Write a list of attributes, as described by a list of attribute records in payload.
0x03	Write attributes undivided	Write attributes, all or nothing mode.
0x04	Write attributes response	Response with a list of status codes, for each parameter.
0x05	Write attributes no response	Write attributes, best effort mode.
0x06	Configure reporting	Only specific attributes support reporting in a cluster. The list of attribute reporting configuration records passed as parameter includes minimal and maximal reporting intervals, the minimum change of the attribute that should trigger reporting. This command may be sent by a client to a server to configure reporting, or by a server to a client to describe its future reporting parameters.

(*Continued*)

Table 7.8 (*Continued*)

Command identifier	Command	Usage
0x07	Configure reporting response	
0x08	Read reporting configuration	Request to get the reporting configuration parameters for one or more attributes
0x09	Read reporting configuration response	
0x0a	Report attributes	Reporting for one or more attributes of a cluster, according to a previously configured reporting relationship (binding)
0x0b	Default response	Basic success/error response
0x0c	Discover attributes	Specifies a starting 16-bit attribute identifier and maximum number of identifiers to report
0x0d	Discover attributes response	
0x0e	Read attributes structured	Used to read only specific index positions (up to 15) for array-type attributes
0x0f	Write attributes structured	Used to write only specific index positions (up to 15) for array-type attributes
0x10	Write attributes structured response	

7.9 ZigBee Application Profiles

An application profile defines a set of messages and attributes standardized for use in a particular context. Application profiles are identified by an application profile ID (0x0000 to 0xFFFF). Each manufacturer can define proprietary messages within its own private application profile, but the ZigBee alliance defines a set of public application profiles, which enable cross-vendor interoperability for the target applications. Profile IDs 0xBF00 to 0xFFFF are reserved for manufacturer specific profiles (MSP), and must be requested from the ZigBee alliance. Profile IDs 0x0000 to 0x7FFF are reserved for public application profiles, and are listed in Table 7.9.

Each ZigBee public profile contains a list of devices identified by a device ID. Each device implements a standard set of features defined by a list of input and output clusters, and a list of attributes (some optional, some mandatory). Within a given public application profile, manufacturers can extend that list by using the "manufacturer-specific extension" field of the ZCL (see Section 7.8.4).

7.9.1 The Home Automation (HA) Application Profile

The HA profile is by far the most widely implemented by manufacturers. While ZigBee does not specify use of nonvolatile memory, the HA profile does: nodes should retain

Table 7.9 ZigBee public application profiles

Public Application Profile		Profile ID	Usage
Home Automation	HA	0x0104	Security, HVAC, LIGHTING CONTROL, ACCESS CONTROL, IRRIGATION. . .
Commercial Building Automation	CBA	0x0105	Security, HVAC, AMR, lighting control, access control
Industrial Plant Monitoring	IPM	0x0101	Asset management, process control, environmental control, energy management
Telecommunications Applications	TA	0x0107	Information delivery in hot zones, public information enquiry, location-based services, remote control (TV, DVD), cell phone
Automatic (Advanced) Metering Initiative or Smart Energy 1	AMI ZSE 1	0x0109	
Personal Home and Hospital (Health) Care	PHHC	0x0108	Patient monitoring, Fitness monitoring.

their configuration (PAN ID, address, group IDs, etc. . . .) even after a power down. This facilitates battery replacement and network restarts after power outages (silent rejoin, see Section 7.3.3).

The HA profile determines the following settings:

- It uses stack profile 0x01 or 0x02.
- Channels 11, 14, 15, 19, 20, 24, 25 are preferred.
- The trust center link key is hardcoded in the profile, the trust center distributes a network key.
- The minimum number of entries of entries in the broadcast transaction table (BTT, see Section 7.4.5.1) is nine.

The standard devices defined by the home automation public application profile are characterized by the mandatory clusters that they must support. Mandatory and optional "common clusters" are defined for all HA devices:

- Server side: mandatory support of basic and identify clusters. Optional support for power configuration, device temperature configuration, alarms, meter, and manufacturer-specific clusters.
- Client side: optional support for meter, and manufacturer-specific clusters.

The HA profile also lists mandatory and optional clusters specific to each device type. Table 7.10 lists some examples.

Table 7.10 Some devices defined by the HA public application profile

	Device name	Device ID	Supported clusters Mc/Ms: mandatory on client or server side, Oc/Os: optional on client or server side
Generic	On/Off switch	0x0103	Ms: OnOff (0x0006) Os: OnOffSwitch Config (0x0007) Oc: Scenes (0x0005) Oc: Groups (0x0004) Oc: Identify (0x0003)
	Range Extender	0x0008	Only common clusters
	Mains Power Outlet	0x0009	Ms: OnOff (0x0006) Ms: Scenes (0x0005) Ms: Groups (0x0004)
Lighting	On/Off Light	0x0100	Ms: OnOff (0x0006) Ms: Scenes (0x0005) Ms: Groups (0x0004)
	DimmableLight	0x0101	Ms: OnOff (0x0006) Ms LevelControl (0x0008) Ms: Scenes (0x0005) Ms: Groups (0x0004) Oc: Occupancy sensing (0x0406)
	Light Sensor	0x0106	Ms: Illuminance measurement (0x0400) Oc: Groups (0x0004)
	DimmerSwitch	0x0104	Mc: OnOff (0x0006) Mc LevelControl (0x0008) Oc: On Off switch configuration (0x0007) Os: Scenes (0x0005) Os: Groups (0x0004)
Closures	Shade	0x0200	Ms: OnOff (0x0006) Ms LevelControl (0x0008) Ms: On Off switch configuration (0x0007) Ms: Scenes (0x0005) Ms: Groups (0x0004)
	Shade Controller	0x0201	Mc:OnOff (0x0006) Mc LevelControl (0x0008) Oc: Shade configuration (0x0100) Oc: Scenes (0x0005) Oc: Groups (0x0004) Oc: Identify (0x0003)

Table 7.10 (*Continued*)

	Device name	Device ID	Supported clusters Mc/Ms: mandatory on client or server side, Oc/Os: optional on client or server side
HVAC	Heating / Cooling unit	0x0300	Ms:OnOff (0x0006) Mc: Thermostat (0x0201) Os: Fan control (0x0202) Os: Level control (0x0008) Os: Groups (0x0004)
	Thermostat	0x0301	Ms: Thermostat (0x0201) Os: Scenes (0x0005) Os: groups (0x0004) Os: Thermostat user interface configuration (0x0204) Os/ Oc: Fan control (0x0202) Os / Oc: Temperature measurement (0x0402) Os / Oc:: Occupancy sensing (0x0406) Os/Oc: Relative humidity measurement (0x0405)
	Temperature sensor	0x0302	Ms: Temperature measurement (0x0402)

7.9.2 ZigBee Smart Energy 1.0 (ZSE or AMI)

ZigBee Smart Energy 1.0 is a public application profile (profile 0x0109), documented in "ZigBee SMART ENERGY PROFILE SPECIFICATION (rev 15, 1/12/2008)". It defines the smart energy devices and clusters required to build an energy-management system (EMS).

The ZigBee Smart Energy 1 (ZSE) public application was defined to enable usage of ZigBee for automatic metering, demand response and prepayment applications required by utilities. The ZSE was defined just before ZigBee decided that its next version would rely on IP, and ZSE was the first application profile to be entirely redesigned in the context of IP, the new specification is called ZigBee Smart Energy 2.0 (see Section 13.4).

ZSE dedicates a secure HAN (use of security is mandatory) to the utility, and defines the communication primitives used between the energy service portal controlled by the utility, and devices located in the end user primitives, such as home displays or load control devices. As the displays and other ZigBee devices may be deployed by end users on their home network using HA, ZSE also defines an extension (the "stub APS") to the ZigBee core specification enabling limited single hop communications between two PANs (the utility PAN and the user home area network).

ZSE defines several new clusters listed in Table 7.11.

7.9.2.1 Security

SE makes a mandatory use of link keys (preconfigured or commissioned), which are otherwise optional in ZigBee. However, a master key is not used or preconfigured in ZigBee SE devices, which do not operate in "high-security" mode.

ZigBee smart energy devices use a network key allocated by the trust center, using the key establishment cluster with a preconfigured trust center link key. The link key is also replaced by the trust center as part of this security bootstrapping process. ZigBee smart energy networks will not generally send keys in the clear.

ZigBee SE envisions that two separate home area networks will be used:

- One network for the exclusive use of the utility company, interconnecting the energy services portal, in-home display(s) and load control devices.
- One separate network for the home owner use, including home automation devices, in-home displays (ihd), a home energy management console, and smart appliances.

The links between the automatic metering infrastructure (AMI) servers and the home area networks may use a combination of non-ZigBee and ZigBee networks. The energy services portal may control a collection of sub-ESPs, in a cascading fashion, so that ZigBee can optionally also be used as a neighborhood area network (NAN).

Since all ZigBee SE devices are configured with multiple keys (link keys and the network key), ZigBee SE defines which cluster uses which key. All the ZigBee SE clusters use application link keys, but most general clusters (e.g., basic cluster, alarm cluster, identify cluster) use the network key.

7.9.2.2 Smart Energy Extensions of ZigBee

ZigBee SE defines a simplified APS layer designed for basic "inter-PAN" communication, that is, communication between a PAN and a device that has not joined. The specification mentions the "refrigerator magnet" with an LCD screen as a target.

Such messages can be sent unicast, broadcast or sent to a group, but without any security.

7.9.2.3 Smart Energy Devices

ZigBee SE defines the following "smart energy" devices:

Energy Service Portal (ESP, Device Id 0x0500)
The energy service portal is a server controlled by the utility company that connects to the metering and energy management devices within the home. The ESP acts as the coordinator and trust center of the network.

The ESP must support the server side of the price, message, demand response/load control and time clusters, and optionally of the complex metering, simple metering and prepayment. It may support the client side of the simple metering, complex metering, price and prepayment clusters.

Metering Device (Device Id 0x0501)
A metering device must support the client side of the metering cluster, optionally of the complex metering cluster. It may support the client side of the metering prepayment, price and message clusters.

In-Premises Display Service (Device Id 0x0502)
The device is designed to be used as a simple user interface, displaying graphs or messages, and able to signal button press events. It must implement the client side of at least one of the price, simple metering and messaging clusters.

Programmable Communicating Thermostat (PCT, Device Id 0x0503)
This device is designed to control heating and cooling systems. It must implement the client side of the Demand response/load control and time clusters. It may implement the client side of the prepayment, price, simple metering and message clusters.

Load Control Device (Device Id 0x0504)
This device is designed to manage the electric consumption of devices in a generic way. It must implement the client side of the demand response/load control and time clusters. It may implement the client side of the price message cluster.

Range Extender Device (Device Id 0x0008)
A device announcing this device Id must be a pure ZigBee router, not supporting any other function.

Smart Appliance Device (Device Id 0x0505)
Smart appliances are able to participate in energy-management policies. They must implement the client side of the price and time clusters, and optionally of the demand response/load control and message clusters.

Prepayment Terminal Device (Device Id 0x0506)
This device is not fully specified yet. It is designed to support advance payment of services, and various display functions.

7.9.2.4 Smart Energy Clusters

Demand Response and Load Control Cluster (0x0701)
Server-Side Commands

Table 7.11 New clusters defined by ZigBee smart energy

Price	0x0700
Demand response and load control	0x0701
Simple metering	0x0702
Message	0x0703
Registration	0x0704
AMI tunneling (complex metering)	0x0705
Prepayment	0x0706

Load control event (0x00). This event is sent to the devices asked to implement a load control action, and specifies the actions required. It includes the following parameters:

- *Issuer Event ID*: unique identifier that will be used to identify future event reports related to this demand response and load control event.
- *Device Class*: bit-encoded field indicating the device classes (end devices actually performing the energy-demand response, as opposed to their ZigBee controllers) needing to participate in an event. The classes defined include HVAC compressors, Strip heaters, water heaters, electric vehicles, and so on.
- *Utility Enrolment Group*: both the utility enrolment group field and the device class must match the target device configured values, otherwise it will ignore the command.
- *Start Time*: UTC Timestamp or 0x00000000 for "now."
- *Criticality Level*: levels 1 to 9 are currently defined. Participation in levels 1 to 6 is voluntary, participation in level 9 is mandatory. Level 1 signals an abnormal percentage of nongreen sources in the delivered energy.
- *Cooling and heating temperature offsets*: offsets, in units of 0.1 °C, to the local temperature setpoint of the thermostat (added for cooling systems, subtracted for heating systems), noncumulative across sequential demand response events. 0xFF when not used.
- *Cooling and heating temperature setpoint*: request to replace the current setpoint by the indicated value between –273 °C and 327 °C in units of 0.01 °C, set to 0x8000 when not used.
- *Average Load Adjustment Percentage*: defines a load offset of -100 to $+100$ points relative to 100%, with a resolution of 1 point. Interpretation is specific to the client implementation. Set to 0x80 when not used.
- *Duty cycle*: a percentage of time between 0 and 100%, 0xFF when not used.
- *Event Control*: flags to indicate whether randomization of the start and end times is required.

Cancel load control event (0x01): cancellation order specifying the issuer event Id, utility enrollment group and device classes concerned. Flag to optionally override the end

randomization settings of the original load control event, and desired cancellation UTC time (0x00000000 for "now").

Cancel all load control events (0x02): same as the cancel load control event command, without the filtering parameters.

Client Side

The client side of the load control/demand response cluster must be able to store at least 3 scheduled events. Events exceeding the storage capacity should be retrieved as soon as possible using command "get scheduled events".

The client side maintains several attributes:

- The utility enrolment group;
- The start randomization period, and the stop randomization period, in minutes;
- The device class bitmask.

It implements two commands:

- **Get Scheduled Events (0x01),** which asks the server side to resend up to a certain number of load control commands scheduled to start at or after the specified UTC time stamp.
- **Report Event Status (0x00),** which reports the time at which the load-control event (identified by its event ID) was effectively executed, the criticality level, the cooling or heating set points, load adjustment percentage, duty cycle that were applied. The event status parameter contains the current state of the load control event: started, completed, user opted in or opted out before or during the event, canceled, superseded, and error conditions. An electronic signature ensures nonrepudiation.

Simple Metering Cluster (0x0702)

Server-Side Attributes

Attribute set identifier	Attributes	
0x00 Reading information set	CurrentSummationDelivered (0x00), CurrentSummationReceived (0x01),	Most recent summed value of energy/gas/water delivered to, or provided by the premises.
	CurrentMaxDemandDelivered (0x02), CurrentMaxDemandReceived (0x03), CurrentMaxDemandDeliveredTime (0x08), CurrentMaxDemandReceivedTime (0x09)	Instant value of maximum rates, and when they were measured.
	DFTSummation (0x04), DailyFreezeTime (0x05),	Snapshot of CurrentSummationDelivered taken at instant DailyFreezeTime

	PowerFactor (0x06), ReadingSnapShotTime (0x07),	Average power factor
0x01 TOU information Set	CurrentTierNSummationReceived CurrentTierNSummationDelivered	Where $N = 1$ to 6, partial summation counters per price tier (defined per period in the time of use schedule or per price tier).
0x02 Meter status	Status (0x00)	Status flags: tamper detected, leaks, etc.
0x03 Formatting	UnitOfMeasure (0x00), Multiplier (0x01), Divisor (0x02), SummationFormatting (0x03), DemandFormatting (0x04), HistoricalConsumptionFormatting (0x05),	Set of attributes used to transform counters to displayable values: units, scaling factors.
0x04 ESP historical consumption	InstantaneousDemand (0x00), CurrentDayConsumptionDelivered (0x01),CurrentDayConsumptionReceived (0x02), PreviousDayConsumptionDelivered (0x03), PreviousDayConsumptionReceived (0x04), CurrentPartialProfileIntervalStartTimeDelivered (0x05), CurrentPartialProfileIntervalStartTimeReceived (0x06), CurrentPartialProfileIntervalValueDelivered (0x07), CurrentPartialProfileIntervalValueReceived (0x08)	Accumulators since midnight for the current day, since the start time of the current profile interval, for the previous day
0x05 Load profile configuration	MaxNumberOfPeriodsDelivered (0x00)	maximum number of intervals the device is capable of returning in one get profile response command.
0x06 Supply Limit	CurrentDemandDelivered (0x00), DemandLimit (0x01), DemandIntegrationPeriod (0x02), NumberOfDemandSubintervals (0x03)	Demand is integrated during the demand integration period, and the integration result is written to currentDemandDelivered at the end of each subinterval.

Server-Side Commands

GetProfileResponse (0x00) returns a number of period accumulators for periods ending before a specified EndTime.

RequestMirror (0x01): request the ESP to mirror metering device data, using RequestMirrorResponse command.

command

RemoveMirror (0x02): request the ESP to remove its mirror of metering device data.

Client-Side Commands

GetProfile (0x00): requests a number of period accumulators for periods ending before a specified EndTime, for received or for delivered quantities.

RequestMirrorResponse (0x01): the ESP informs a sleepy metering device it has the ability to store and mirror its data, which should be sent to the indicated endpointID.

RemoveMirror (0x02): the ESP no longer has the ability to mirror data.

Price Cluster (0x0700)

Server-Side Attributes

6 price tiers labels are defined, by attributes "TierXpriceLabel" (0x0000 to 0x0006). The price tiers are defined by command publish price.

Client-Side Commands

- **getCurrentPrice (0x00)**: initiates a publish price command for the current time.
- **getScheduledPrices (0x01)**: initiates a publish price command for all currently scheduled times after the provided time stamp (up to a maximum number of scheduled times also specified in the command).

Server-Side Commands

Publish price (0x00): this command defines a new price tier and is sent in response to a getCurrentPrice or getScheduledPrices command. It contains the following subfields:

- *ProviderID:* unique Id of the commodity provider;
- *Rate Label:* 12 character UTF8 string related to current rate;
- *Issuer Event ID:* unique identifier for this pricing information, must increase when newer prices are published for the same period;
- *Current Time:* UTC time of the sending node;
- *Unit of measure:* 8-bit field defining the commodity and unit of measure;
- *Currency*
- *Price Trailing Digit and Price tier:* bit field indicating the price tier for this rate, and the position of the decimal point in the published price;

- *Number of price tiers and Register tier:* number of price tiers in use (0 to 6), for the current tier, indication of which CurrenttierXSummationDelivered accumulator relates to the current price tier;
- *StartTime:* UTC start time of the current rate signal, 0x00000000 means "now";
- *Duration in minutes:* validity of this price signal. 0xffffffff means "until changed";
- *Price:* price per base unit;
- *PriceRatio* (optional): ratio to the "normal" tariff;
- *Generation Price* (optional): price for commodity received from the premises;
- *AlternateCostDelivered*, Alternate cost unit and alternate cost trailing digit: cost using an alternate measure, for example, grams of CO_2 per kWh.

Messaging Cluster (0x0703)
Server-Side Commands
DisplayMessage (0x00): specifies the level of importance of the message, whether a confirmation is required, and the number of minutes the message must be displayed.
CancelMessage (0x01)

Client-Side Commands
GetLastMessage (0x00): request to send a DisplayMessage command
MessageConfirmation (0x01): used to acknowledge a message, provides a timestamp.

Smart Energy Tunneling (Complex Metering) Cluster (0x0704)
Not defined yet, this cluster is a placeholder for future tunneling mechanisms for more sophisticated metering protocols, for example, C.12 or DLMS/COSEM.

Prepayment Cluster (0x0705)
Not defined yet.

7.10 The ZigBee Gateway Specification for Network Devices

The ZigBee Gateway specification for network devices, Version 1.0, was released on July 27th 2011. It had been a work in progress since 2007. This specification defines several possible communication protocols between ZigBee gateway devices (ZGD) that implement a bidirectional interface between ZigBee 1.0 PANs and IP networks, and IP host applications (IPHA).

The ZGD provides access to:

- ZCL operations: read and write attributes, configure notifications and report events;
- ZDO operations;
- Macro operations simplifying network and service discovery;
- Endpoint management;

- Its own ZigBee information bases (AIB, NIB, and PIB attributes);
- Network startup and join functions on behalf of the IPHA;
- Security material configuration and operations.

The communication between ZGDs and IPHAs is bidirectional, both act as client and server. The IPHA calls procedures implemented by the ZGD, and the ZGD calls event handlers of the IPHA. These RPC functions have been specified in an abstract, protocol-independent request-response format that needs to be complemented by a protocol binding specification.

Some ZGD procedures operate only in blocking mode, while other procedures offer a choice of blocking or nonblocking mode (such nonblocking procedures provide a *CallbackDestination*, which is an IPHA callback URL or an empty string if the IPHA uses polling). All IPHA event handlers are nonblocking.

ZigBee PAN messages received by the ZGD are processed by one or more ZGD callback handlers, which decode and feed them to requesting IPHAs. The configuration of these callback handlers is persistent across ZGD power cycles.

Version 1.0 of the ZigBee gateway specification proposes three possible network bindings implementing the ZGD to IPHA remote procedure calls: a SOAP binding, a REST binding, and a GRIP (gateway remote interface protocol) binding.

7.10.1 The ZGD

The ZGD itself is a ZigBee device. As such it should support the ZigBee commissioning cluster, as well as mandatory ZDO client and server clusters.

The ZGD functions belong to several categories: gateway management object, ZigBee device profile, ZigBee cluster library, application support sublayer, inter-PAN, and network layer.

An IPHA can access:

- APS commands to read and modify the ZGD configuration by using a set of APS functions. It can read or set descriptors (ConfigureNodeDescriptor, GetNodeDescriptor, GetNodePowerDescriptor, ConfigureUserDescriptor, GetUserDescriptor), manage local ZGD endpoints (ConfigureEndpoint, ClearEndpoint), send and receive APS frames (SendAPSMessage / NotifyAPSMessageEvent), manipulate local groups (AddGroup, RemoveGroup, RemoveAllGroups, GetGroupList), and read local bindings (GetBindingList).
- Network layer commands to get access to the Network layer management entity (NLME) of the ZGD (FormNetworkProcedure, FormNetworkEvent, StartRouter, StartRouterEvent, Join, JoinEvent, Leave, LeaveEvent, Reset, ResetEvent, DiscoverNetworks, DiscoverNetworksEvent, PerformEnergyScan, PerformEnergyScanEvent,

NetworkStatusEvent, PerformRouteDiscovery, PerformRouteDiscoveryEvent, SendNWKMessage, NotifyNWKMessageEvent).

An IPHA can send arbitrary ZDP frames by using the SendZDPCommand ZGD function and receive arbitrary ZDP frames by implementing a NotifyZDPEvent event handler.

The ZGD also provides access to the ZCL. An IPHA can send and receive arbitrary ZCL commands by using the SendZCLCommand ZGD function and implementing the NotifyZCLEvent event handler.

In addition the ZGD supports the specific gateway management object (GMO). The GMO provides functions:

- Enabling the IPHA to retrieve the ZGD version and feature set (GetVersion).
- Enabling read/write of ZGD information base (Get/Set).
- Event CallBack management (CreateCallBack, DeleteCallBack, ListCallBacks) and polling by IPHA (PollCallBack, PollResults, UpdateTimeOut).
- Facilitating network and service discovery (StartNodeDiscovery, NodeDiscoveryEvent, NodeLeaveEvent, ReadNodeCache, StartServiceDiscovery, ServiceDiscoveryEvent, GetServiceDescriptor, ServiceDiscoveryEvent, ReadServiceCache).
- Controlling the Gateway insertion into the PAN (StartGatewayDevice, StartGatewayDeviceEvent, ConfigureStartupAttributeSet, ReadStartupAttributeSet).
- Managing address aliases (CreateAliasAddress, DeleteAliasAddress, ListAddresses): In order to facilitate the mapping of addresses between the PAN network (short 16-bit addresses may change in the PAN) and the IPHA, the ZGD maintains an address alias table where IPHAs can associate their own alias to any 64-bit address. The IPHA or ZGD may use special code 0x10 in the Destination/source address mode[2] of ZDP messages to indicate use of an alias address: the ZGD will translate to 64-bit or 16-bit addresses as required.

7.10.2 GRIP Binding

The gateway remote interface protocol (GRIP) is a lightweight request/response protocol built over SCoP.

The secured connection protocol (SCoP) was designed in the context of the ZigBee bridge device specification as an IP tunneling protocol between ZigBee PAN gateways. It provides support for datagram and stream-oriented communications as well as fragmentation by means of a socket-like

[2] **APSDE-DATA.request DstAddrMode parameter values :** 0x00 = DstAddress and DstEndpoint not Present; 0x01 = 16-bit group address for DstAddress; DstEndpoint not present; 0x02 = 16-bit address for DstAddress and DstEndpoint present; 0x03 = 64-bit extended address for DstAddress and DstEndpoint present; 0x04 – 0xff = reserved.

interface provided by the SCoP data entity (SCDE), on top of UDP, TCP or TLS. SCoP security leverages CCM* (SCoP security service SCSS). Management services are provided by the SCoP management entity (SCME).

SCoP messages target a specific service (identified by a service identifier code), on the target gateway. Currently the supported services are SCoP service (hello, goodbye, keepalive commands), bridge service and GRIP service.

The GRIP API provides a GRIDE-DATA request that must be provided with target information (DestIPVersion, DestIPAddress, DestPort), desired transport parameters (TransportMode, SecurityLevel) and application-level parameters (target FunctionDomain, FunctionCategory, ManufacturerCode, FunctionIdentifier, FunctionParameters).

Requests are provided to the target GRIP application by means of the GRIDE-DATA.indication API. Responses are returned to the querying GRIP application by means of the GRIDE-DATA.response API, and specifies a Status code as well as a function result (an octet string).

The GRIP binding specifies specific GRIP function codes and parameter formats to transport all the generic ZGD procedures defined in Section 7.10.1. The following function categories are defined: Manufacturer specific, GMO, ZDP, ZCL, APS, INTERPAN, NWK. Parameter encoding uses ASN.1 distinguished encoding rules (DER).[3]

7.10.3 SOAP Binding

The SOAP binding specifies a standard web services interface by means of a WSDL document. All the generic ZGD procedures defined in Section 7.10.1 are covered.

7.10.4 REST Binding

Due to the resource oriented design pattern of REST, the REST binding for the ZDG functional protocol is not as straightforward as the GRIP or SOAP bindings.

A set of resources are defined to represent the ZGD (Figure 7.8), each ZigBee network and each ZigBee node (Figures 7.9 and 7.10), each resource is addressable with an

[3] ISO 8825, 1998: Information Technology. ASN.1 Encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). International Standard ITU-T X690 (1997) | ISO/IEC 8825-1:1998.

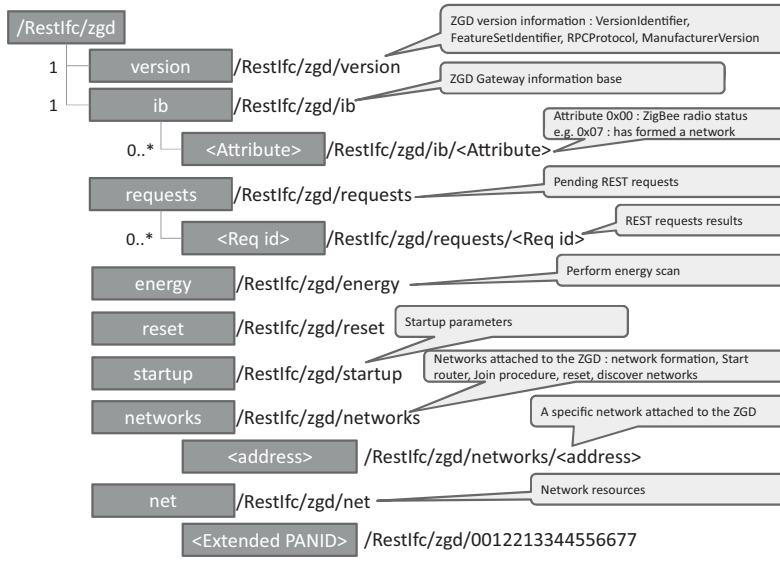

Figure 7.8 ZGD REST resources overview.

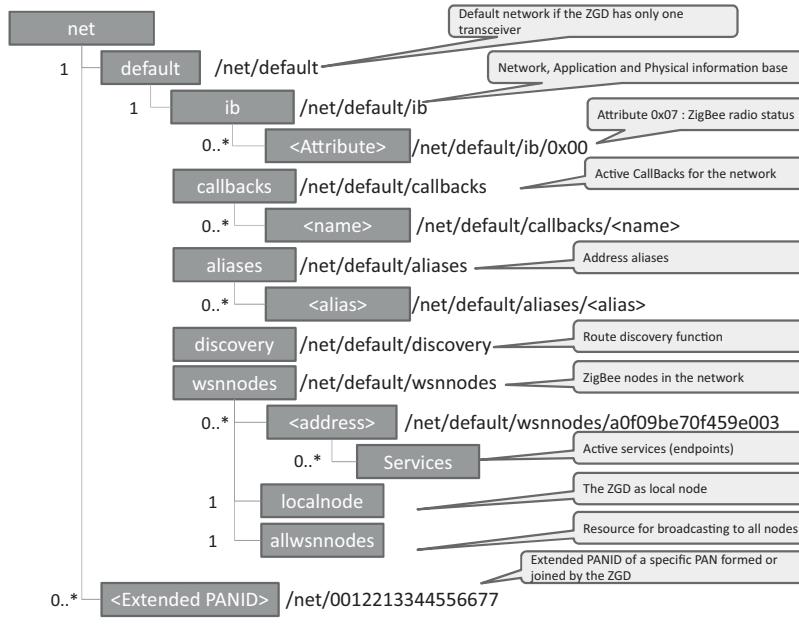

Figure 7.9 Overview of network and node resources.

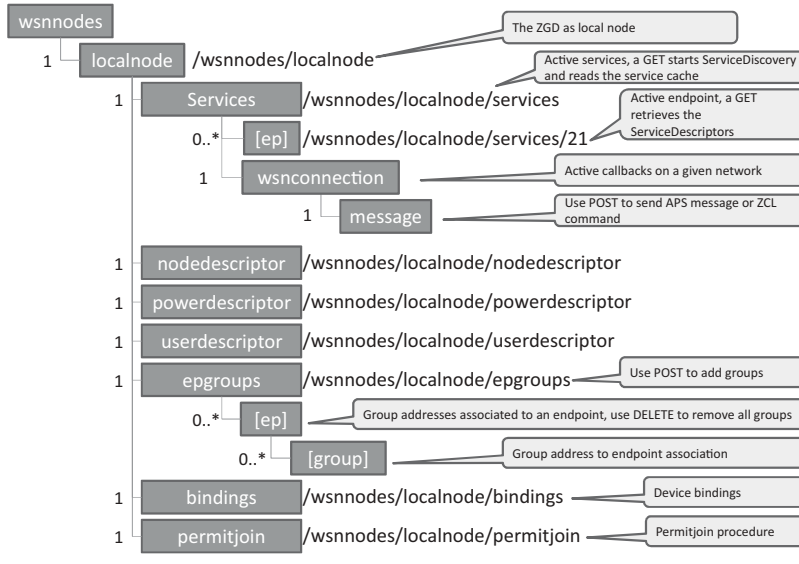

Figure 7.10 Overview of ZigBee node resources.

URL. Each ZGD function is implemented as a state transfer between the IPHA and the appropriate resource: For instance, the GetVersion function, used by the IPHA to discover the features of a ZGD, is implemented as a read operation on the "version" resource of the ZGD.

Asynchronous notifications (responses and events) use a special resource hosted by the IPHA. The IPHA declares the URI of this resource to the ZGD using the CreateCallBack REST operation, and can also specify a specific callback URI in each REST request supporting the CallBackDestination parameter.

The resources are represented by XML documents, specified as part of the REST binding.

7.10.5 Example IPHA–ZGD Interaction Using the REST Binding

The IPHA may read any resource exposed by the ZGD, for instance

```
HTTP GET:
http://''ZGD_IP_Addr:ZGD_PORT''/RestIfc/zgd/version
```

And the ZGD responds with the requested resource representation

```
<tns:Info xmlns:tns=''http://www. zigbee.org/GWGSchema'' >
    <tns:Status>
       <tns :code>0x00</tns :code>
    </tns:Status>
    <tns:Detail>
       <tns:Version>
          <tns:VersionIdentifier>0x01</tns:VersionIdentifier>
          <tns:FeatureSetIdentifier>0x00</tns:
             FeatureSetIdentifier>
          <tns:RPCProtocol>0x0004</tns:RPCProtocol>
          <tns:ManufacturerVersion>1.1</tns:ManufacturerVersion>
          </tns:Version>
    <tns :Detail>
</tns:Info>
```

In this example we suppose that the IPHA needs to allocate a dedicated endpoint on the ZGD. It therefore begins by registering a new endpoint on the ZGD. In the following example we use a synchronous transaction. The IPHA could have use an asynchronous method by specifying a callback URI in parameter *CallbackDestination* of request URI sent to the ZGD.

HTTP POST: HTTP POST: http://''ZGD_IP_Addr:ZGD_PORT''/localnode/services

```
<?xml version=''1.0'' encoding=''UTF-8''?>
<tns:SimpleDescriptor
xmlns:tns=''http://www. zigbee.org/GWGSchema'' >
<tns:ApplicationProfileIdentifier>0x0104</tns:ApplicationProfile
Identifier>
<tns:ApplicationDeviceIdentifier>0x0002</tns:ApplicationDevice
Identifier>
<tns:ApplicationDeviceVersion>0x00</tns:ApplicationDevice
Version>
<tns:ApplicationInputCluster>0x0000</tns:ApplicationInput
Cluster>
<tns:ApplicationInputCluster>0x0003</tns:ApplicationInput
Cluster>
<tns:ApplicationInputCluster>0x0004</tns:ApplicationInput
Cluster>
<tns:ApplicationInputCluster>0x0005</tns:ApplicationInput
Cluster>
<tns:ApplicationInputCluster>0x0006</tns:ApplicationInput
Cluster>
<tns:ApplicationOutputCluster>0x0003</tns:ApplicationOutput
Cluster>
</tns:SimpleDescriptor>
```

The IPHA did not request a specific endpoint, therefore it will be allocated by the ZGD. If the request succeeds, the IPHA will receive a response similar to the following:

```
<tns:Info xmlns:tns=''http://www. zigbee.org/GWGSchema'' >
    <tns:Status>
        <tns :code>0x00</tns :code>
    </tns:Status>
    <tns:Detail>
        <tns:endpoint>0x23</tns:endpoint>
    <tns :Detail>
</tns:Info>
```

This is an example of a synchronous response. When using an asynchronous mode, the response would include a request identifier attribute allocated by the ZGD.

The IPHA has the option of registering a callback function bound to this endpoint or to all endpoints, by invoking an HTTP POST request to [NwkRootURI]/callbacks that specifies filters which will be used by the ZGD to identify messages that will be notified to the IPHA.[4]

The IPHA may then instruct the ZGD to send any command. The endpoint identifier used in the URI is 35 (0x23). For instance an APSMessage:

```
POST [NwkRootURI]/localnode/services/35/wsnconnection/message
<?xml version=''1.0'' encoding=''UTF-8''?>
<tns:APSMessage xmlns:tns=''http://www.zigbee.org/GWGSchema''>
    <tns:DestinationAddress>
        <tns:NetworkAddress>0x0001</tns:NetworkAddress>
    </tns:DestinationAddress>
    <tns:DestinationEndpoint>0x02</tns:DestinationEndpoint>
    <tns:SourceEndpoint>0x23</tns:SourceEndpoint>
    <tns:ProfileID>0x0104</tns:ProfileID>
    <tns:ClusterID>0x0000</tns:ClusterID>
    <tns:Data>0102030405060708090a0b0c0d0e0f</tns:Data>
    <tns:TxOptions>
    <tns:SecurityEnabled>true</tns:SecurityEnabled>
    <tns:UseNetworkKey>true</tns:UseNetworkKey>
    <tns:Acknowledged>true</tns:Acknowledged>
    <tns:PermitFragmentation>true</tns:PermitFragmentation>
    </tns:TxOptions>
    <tns:Radius>3</tns:Radius>
</tns:APSMessage>
```

[4] Callbacks may also be registered using resources [NwkRootURI]/localnode/services/[ep]/wsnconnection or [NwkRootURI]/localnode/allservices/wsnconnection, in which case filters are preset for all APS messages matching the endpoint id.

If the APS command is successful, the IPHA will receive the following response from the ZGD:

```
<tns:Info xmlns:tns=''http://www. zigbee.org/GWGSchema'' >
    <tns:Status>
        <tns :code>0x00</tns :code>
    </tns:Status>
    <tns:Detail>
        <tns: APSMessageResult xmlns:tns=''http://www. zigbee.
        org/GWGSchema''>
            <tns:ConfirmStatus>0x00</tns:ConfirmStatus>
            <tns:TxTime>0x01234567</ts:TxTime>
        </tns:APSMessageResult>
    <tns :Detail>
</tns:Info>
```