life.augmented

AN5627

Application note

STM32WB Series ZigBee commissioning guide

Introduction

The objective of this document is to describe the commissioning process for the Zigbee® functionality on the STM32WB Series microcontrollers.

The guide covers the following subjects:

- Centralized network
- Distributed network
- ZCL commissioning cluster
- Pre-configured startup
- Touchlink commissioning
- Finding and Binding process

This will allow a STM32WB Series microcontroller to join the required network using the Zigbee® protocol.

1 General information

This document applies to the STM32WB Series dual-core Arm®-based Series microprocessor.

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Glossary

Table 1. Glossary

Term	Definition
APS	Application support sublayer
PAN	Personal area network
PANID	Personal area network identification
InterPAN	Process by which devices on different Zigbee®networks communicate with each other
TC	Trust center
TCLK	Trace clock, refer to *Multiprotocol wireless 32-bit MCU Arm®-based Cortex®-M4 with FPU, Bluetooth® Low-Energy and 802.15.4 radio solution* (RM0434)
Touchlink	See Section 6
ZC	Zigbee® coordinator
ZCL	Zigbee® cluster library
ZDO	Zigbee® device objects

1.2 Reference documents

Table 2. Reference documents

Reference	Document
[1]	Zigbee R22 Specification
[2]	Base Device Behavior Specification (13-0402)

2 Centralized network

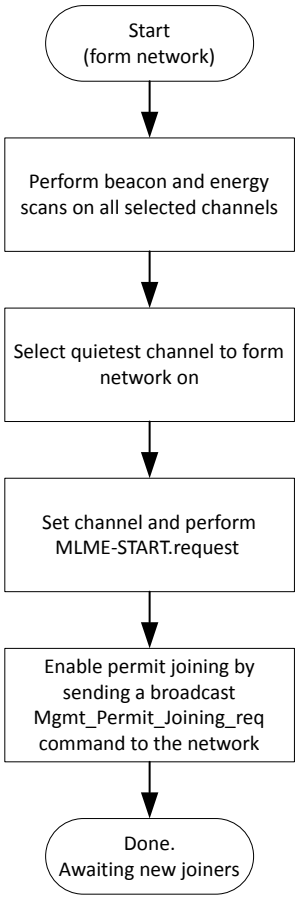
2.1 Forming a Network

This section must be read in conjunction with Zigbee R22 Specification [1] - Section 3.6.1.1.

2.1.1 Network forming flowchart

The following flow chart gives a high level overview of the network formation process:

Figure 1. Flowchart describing Zigbee® network formation



2.1.2 Network forming configuration

1. Zigbee® startup configuration data structure:

```
struct ZbStartupT config;
```

2. Initialize the structure with default configuration:

```
ZbStartupConfigGetProDefaults(&config);
```

3. Select to form a new network:

```
config.startupControl = ZbStartTypeForm;
```

4. Optionally selects an extended PANID to use. If set to zero, then the local device extended address is used instead.

```
config.extendedPanId = 0x1234567812345678ULL;
```

5. Configure the pre-configured link key, or set this parameter to 0x00 if configuring a trust center (TC) and using unique link keys (also known as install codes).

```
memcpy(config.security.preconfiguredLinkKey, sec_key_ha, ZB_SEC_KEYSIZE);
```

2.1.3 Network forming startup

Call the Zigbee® stack startup function. The callback function provided is called once the startup is completed regardless of whether it succeeded or failed.

```
ZbStartup(zb, &config, callback, arg);
```

2.1.4 Network permitting devices to join

```
struct ZbZdoPermitJoinReqT req;

req.destAddr = ZB_NWK_ADDR_BCAST_ROUTERS;
req.duration = 180;
ZbZdoPermitJoinReq(zb, &req, callback, arg);
```

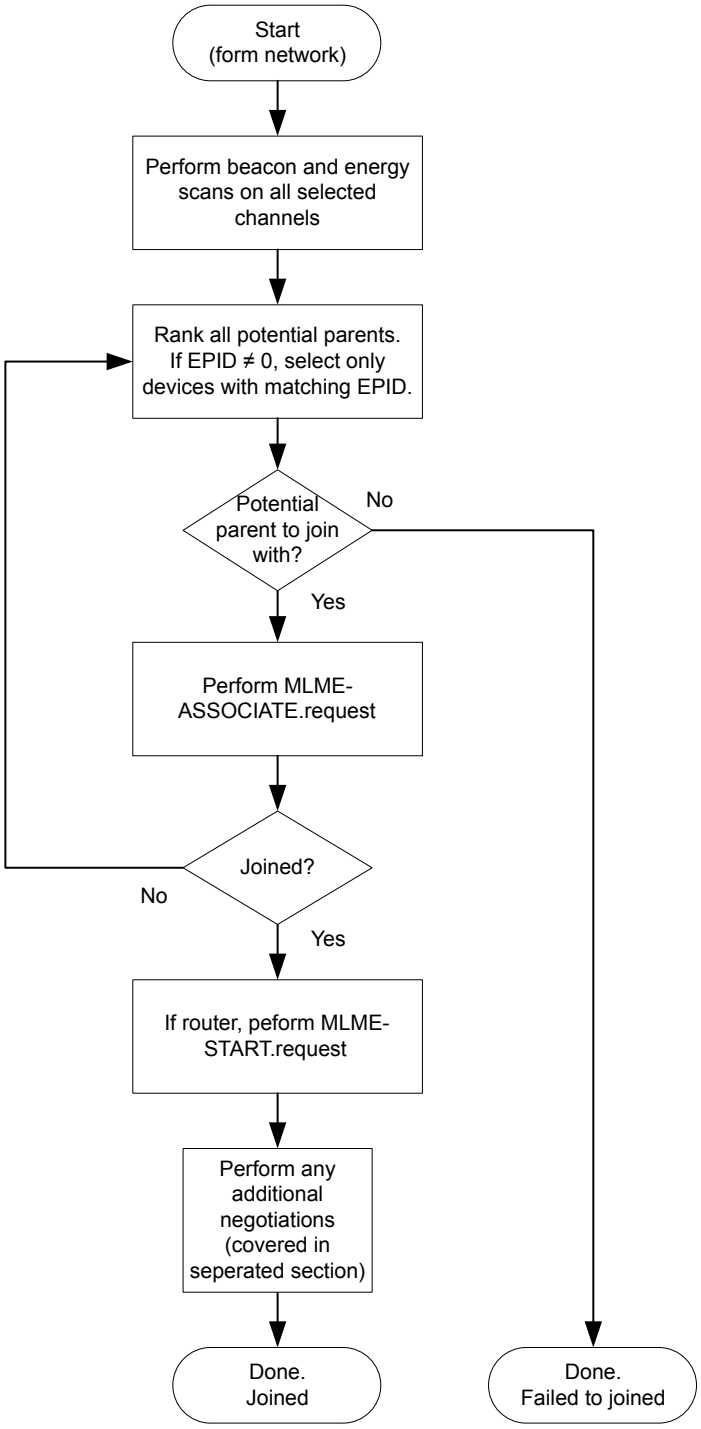
2.2 Joining a network

This section must be read in conjunction with Zigbee R22 Specification [1] - Section 3.6.1.3.

2.2.1 Joining a network flowchart

The following flow chart gives a high level overview of the network joining process:

Figure 2. Flowchart describing joining a Zigbee® network



2.2.2 Joining a network configuration

1. Zigbee® startup configuration data structure:

```
struct ZbStartupT config;
```

2. Initialize the structure with default configuration:

```
ZbStartupConfigGetProDefaults(&config);
```

3. Select to join new network:

```
config.startupControl = ZbStartTypeJoin;
```

4. Optionally selects an extended PANID to use. If set to zero, the device attempts to join any open network available:

```
config.extendedPanId = 0ULL;
```

5. Pre-configured link key:

```
memcpy(config.security.preconfiguredLinkKey, sec_key_ha, ZB_SEC_KEYSIZE);
```

2.2.3 Joining a network startup

Call the Zigbee® stack startup function. The callback function provided is called once the startup is completed regardless of whether it succeeded or failed.

```
ZbStartup(zb, &config, callback, arg);
```

2.3 Pre-configured link keys

This section must be read in conjunction with Zigbee R22 Specification [1] - Section 4.2.1.2.1.

Pre-configured link keys are programmed to the device before it joins a network. This is done at the time of manufacture or during the application installation.

2.3.1 Global link key

A global link key involves all the devices using the same link key for the joining process, usually to decrypt the APS transport key sent from the trust center (or parent in a distributed network) which contains the network key.

For example, the default link key used during Zigbee® stack testing is:

```
"ZigBeeAlliance09"
```

Or:

```
5a:69:67:42:65:65:41:6c:6c:69:61:6e:63:65:30:39
```

Global link keys are used because all the network devices can be programmed at the time of production with identical keys. This means the installer does not need to perform any further configuration when joining devices to the network.

2.3.2 Install code link key

An install code link key is a link key that is unique to the joining device. These keys are only used in centralized networks with a trust center. Before the device joins the network, the trust center has the device install code link key added to its key table. During the joining process, when the trust center sends APS encrypted packets to the joining device, such as the APS transport key containing the network key, the trust center uses the install code link key for encryption.

Only the trust center and the joining device will be able to decrypt the APS packet payload. As such, install code link keys keep the network more secure as compared to using global link keys. If an attacker is able to gain a network global link key, the network key can be obtained by listening for the transport key sent during the device joining process.

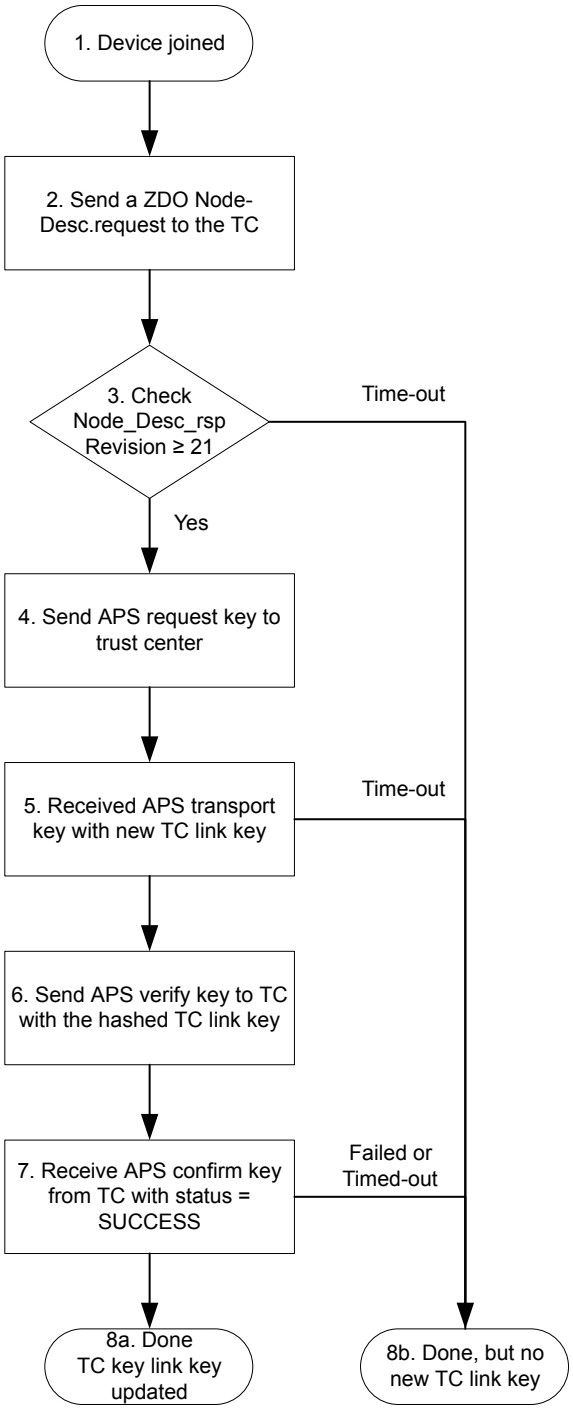
2.4 Trust center link key negotiation

After a device joins a centralized network, it negotiates a unique link key with the trust center. This key is only shared between the joiner device and the trust center, so other devices in the network or devices listening to network traffic are not able to decrypt the APS payloads encrypted with the negotiated link key.

2.4.1 Trust center link key negotiation flowchart

The figure below illustrates the trust center link key negotiation process.

Figure 3. Flowchart describing trust center link key negotiation



The steps illustrated in the figure above are detailed in the following list:

1. The device has joined, received the network key from the TC and the device has sent a `ZDO Device-Annce` message.
2. The device sends a `ZDO Node-Descriptor.request` to the TC to query for the TC stack revision number.

3. If the ZDO `Node-Descriptor.response` is received and indicates a stack revision of at least 21, then the TC supports TCLK negotiation.
4. The device sends an APS request key to the TC. Upon reception, the TC generates a new link key to share between the TC and the device. The TC generates an APS transport key command with the new link key and sends it to the device.
5. The device receives the new TC link key and updates it's key table.
6. The device generates an APS verify key command containing a hash of the new TC link key and sends it to the TC. Upon receiving the APS verify key, the TC verifies the hashed key and sends an APS confirm key command to the device.
7. The device receives the APS confirm key and checks the status.
8. There are two possible outcomes to this process:
   a. The `ZbStartup` callback is called, indicating to the application that the join procedure has been successful.
   b. The `ZbStartup` callback is called, indicating to the application that the join procedure has failed.

2.5 Centralized network APIs

Trust center link key negotiation is enabled by default in the stack. However, here are the steps to enable/disable it before calling `ZbStartup`. The API sample is given in the code below.

```
/* Enable or disable the Trust Center Link Key Negotiation */
uint8_t val = 1;
ZbBdbSet(zb, ZB_BDB_TrustCenterRequiresKeyExchange, &val, sizeof(val));

/* Control the Trust Center Link Key Method.
 * Set to 0x00 for standard APS Trust Center Link Key negotiation.
 * Set to 0x01 for Smart Energy Certificate Based Key Exchange (CBKE) */
uint8_t val = BDB_LINKKEY_EXCHANGE_METHOD_APS;
ZbBdbSet(zb, ZB_BDB_TCLinkKeyExchangeMethod, &val, sizeof(val));
```

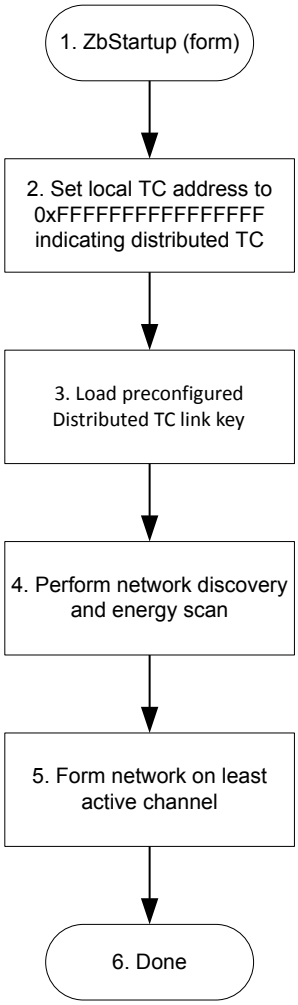
3 Distributed network

A distributed network is one that has no centralized coordinator or trust center. Each router within the network is capable of providing a joining device with the network key after a MAC association. The APS traffic is encrypted with a known global link key that all devices are pre-programed with the silicon manufacturer settings or during commissioning.

3.1 Distributed network flowchart

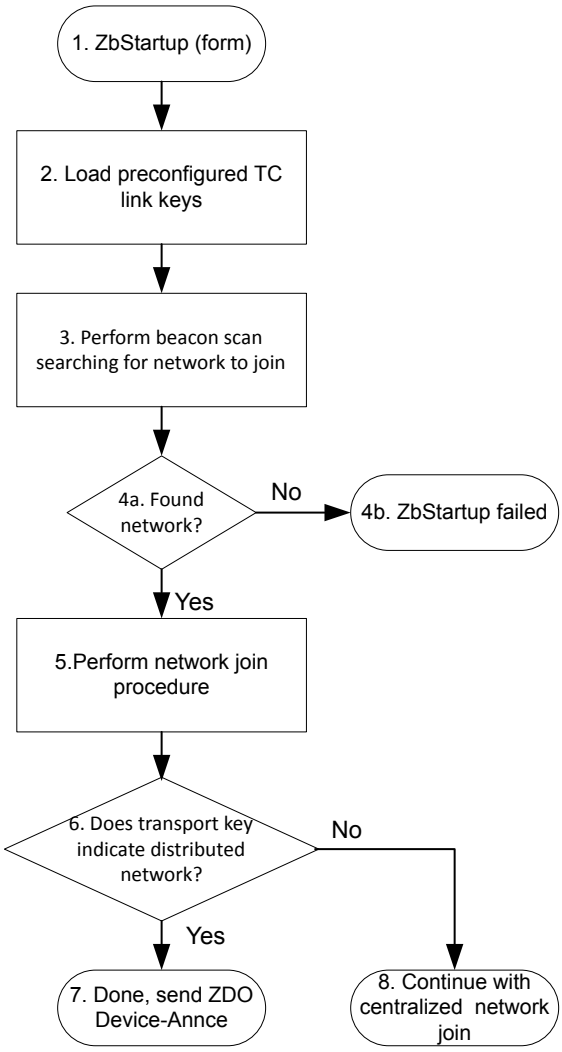
The figure below illustrates the Zigbee® distributed network creation.

Figure 4. Flowchart describing device forming a Zigbee® distributed network



The procedure to form a distributed network is the same as a centralized network, except that the trust center address is configured to 0xFFFFFFFFFFFFFFFF in the `ZbStartup` configuration. The preconfigured distributed link key is also set, which is the global link key that all devices use to join to the network. Since there is no centralized coordinator (or TC), all router parent devices act as trust centers and send a transport key, encrypted with the distributed link key, containing the network key after a device joins. The Zigbee® distributed network joining process is illustrated in the figure below.

Figure 5. Flowchart describing device joining a Zigbee® distributed network



1. Application calls `ZclStatusCodeT` to join a new network.
2. The pre-configured distributed link key provided to `ZclStatusCodeT` configuration, is added to the stack. It will be used to decrypt the transport key sent by the parent.
3. The device scans the channels looking for a network to join.
4. If a network has been found, move on to the normal Zigbee® join procedure, otherwise startup has failed (step 4b)
5. Device tries to join network.
6. After joining, the transport key must have an extended source address of 0xFFFFFFFFFFFFFFFF, indicating the device has joined a distributed TC network.
   - If not, then continue with the normal Zigbee® join procedure (step 8).
   - If so, then the device has joined the network and has the network key needed to communicate on the network.
7. Join complete. Send a `ZDO Device-Annce` to the network.

   else
8. Continue with the centralized network connection.

3.2 Distributed network APIs

Configure the following `ZbStartupT` parameters when forming or joining a distributed network.

3.2.1 Distributed network trust center address

```
/* Configure the Trust Center to be distributed (0xffffffffffffffff) */
config.security.trustCenterAddress = ZB_DISTRIBUTED_TC_ADDR;
```

3.2.2 Distributed network pre-configured distributed link key

```
/* Configure the Preconfigured Distributed Link Key for the network.
* For testing, use the "Uncertified Distributed Key Key", which is
* d0:d1:d2:d3:d4:d5:d6:d7:d8:d9:da:db:dc:dd:de:df */
memcpy(config.security.distributedGlobalKey, sec_key_distrib_uncert,ZB_SEC_KEYSIZE);
```

3.2.3 Distributed network startup

Call the Zigbee® stack startup function. The callback function provided will be called once the startup has completed whether it succeeded or failed.

```
ZbStartup(zb, &config, callback, arg);
```

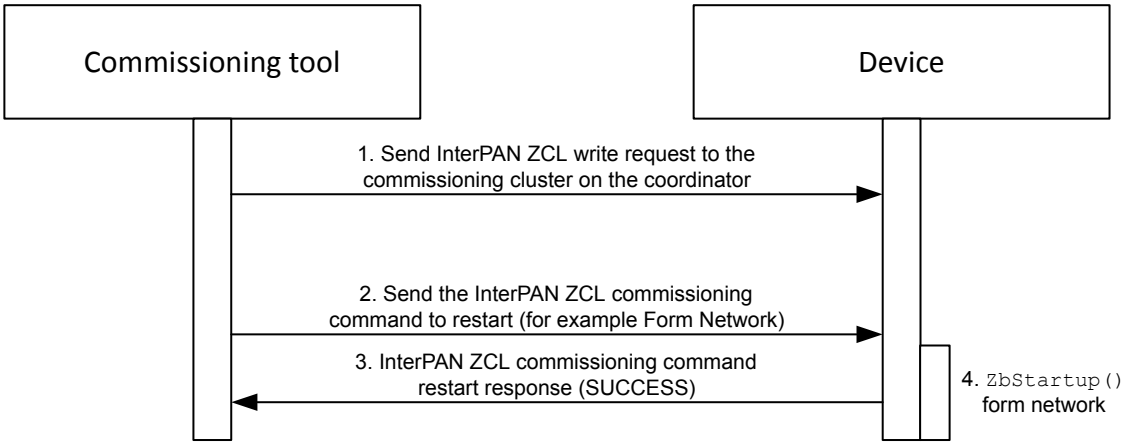
4 ZCL commissioning cluster

The commissioning cluster is used to commission a device during the installation process. The joining device is turned on, and a commissioning tool is then used to transmit the startup parameters necessary to form or join a network. Once the parameters are configured to the joining device, the commissioning tool sends the device a command to start the form or join process.

4.1 InterPAN commissioning sequence chart

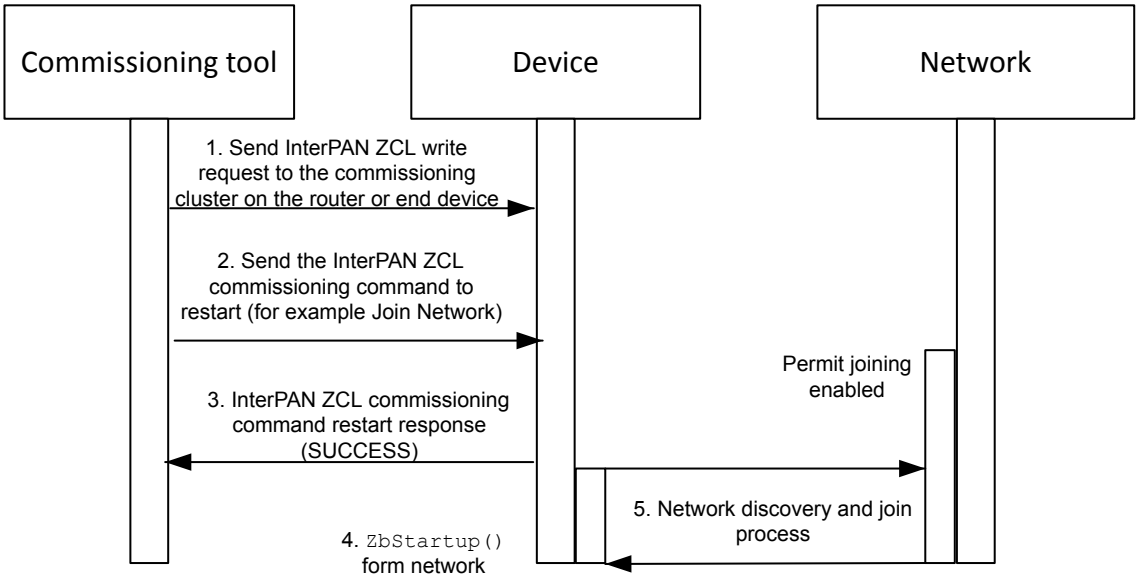
The InterPAN network creation process is outlined in the figure below and includes the necessary messages.

Figure 6. InterPAN commissioning messages



1. The commissioning tool sends "ZCL Write Attribute" commands to the device to be commissioned in order to configure the Zigbee® Startup Attribute Set. One of the attributes sets the `StartupControl` to form a network.
2. The commissioning tool sends the command to restart the device using the "current set of startup parameters".
3. The device responds with the "Restart Response" command and status of SUCCESS.
4. The device performs the "Network Formation" procedure and forms a new network.

Figure 7. InterPAN Commissioning Messages: joining a network

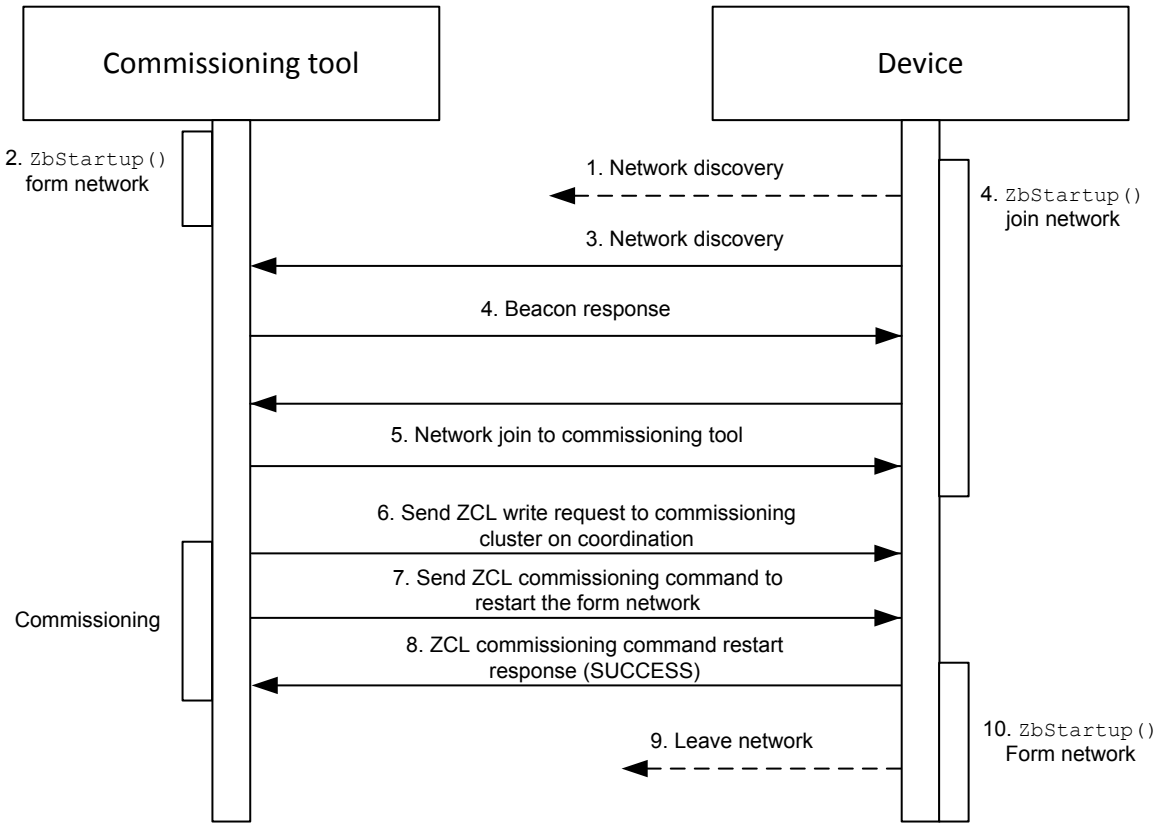


The steps (1-3) are the same as with the commissioning tool configuring a device to form a network, but in this case the `StartupControl` is set to join a network. After the device receives the restart request command, it will start the "Network Join" procedure (steps 4-5).

4.2 Temporary network commissioning sequence chart

A device is able to join a network on a temporary basis. The process is defined in the figure below.

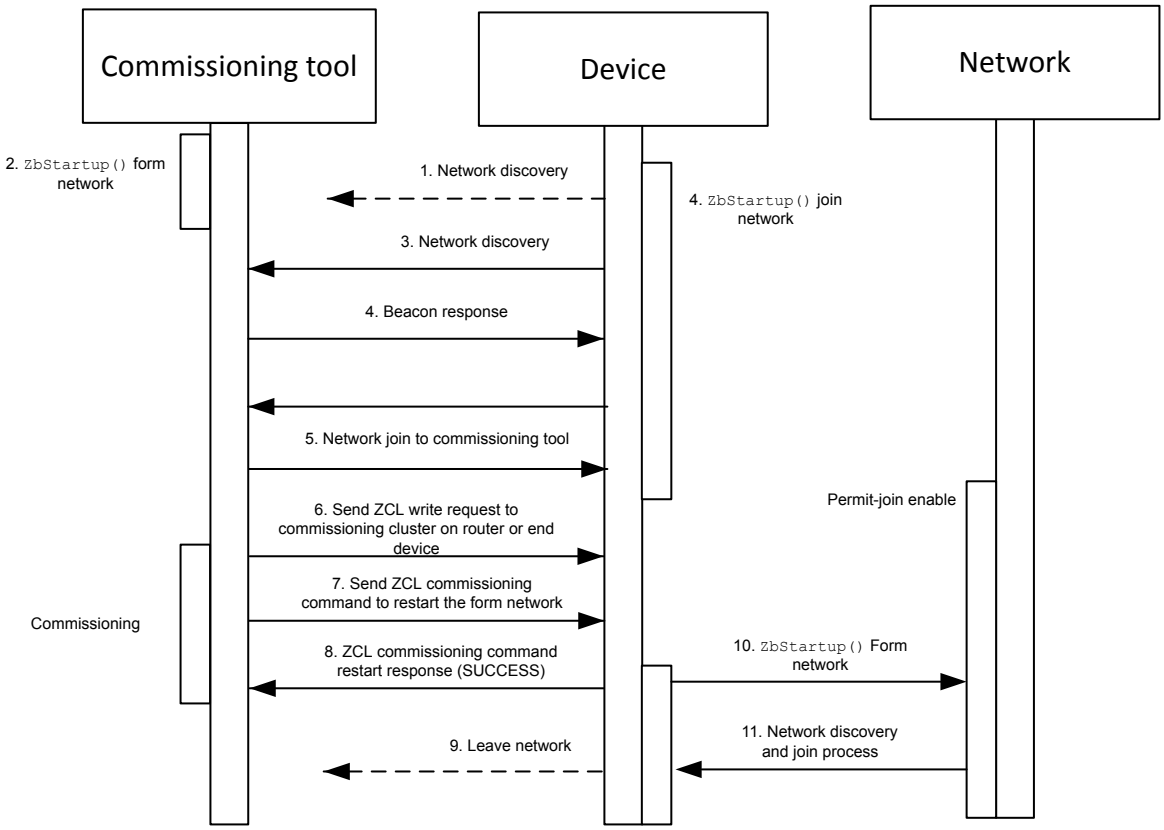
Figure 8. Commissioning tool network formation: tells device to form new network



The commissioning tool definition is details below.

1. The device is trying to join the commissioning tool network, either automatically or through user intervention (push button). At this point, the commissioning tool network has not been formed yet.
2. The commissioning tool forms a temporary network for the device to join to. The default EPID to use is 00-50-C2-77-10-00-00-00, so any device is able to filter for the commissioning tool network from any other nearby networks.
3. The device tries to discover the commissioning tool network again.
4. The commissioning tool responds with a beacon.
5. The device joins the commissioning tool network.
6. The commissioning tool sends ZCL write attribute commands to the device to be commissioned in order to configure the Zigbee® startup attribute set. One of the attributes sets the `StartupControl` to form a network.
7. The commissioning tool sends the command the device to restart using the "current set of startup parameters."
8. The device responds with the restart response command and status of "SUCCESS".
9. Device leaves the commissioning tool network.
10. The device performs the network formation procedure and forms a new network.

Figure 9. **Commissioning Tool Forms Network; Tells Device to Join Desired Network**



The steps (1-7) are the same as with the commissioning tool configuring a device to form a network, but in this case the `StartupControl` is set to join a network. After the device receives the restart request command, it will start the network join procedure (steps 8-10).

4.3 Device commissioning server

The commissioning server is used by the device being commissioned. The following code extract describes how to allocate and instantiate the ZCL commissioning server cluster.

```
struct ZbZclClusterT *cluster;
struct ZbZclCommissionServerCallbacksT callbacks;

/* Configure the Commissioning Cluster function callback handlers for received Commissioning
Client commands. */
memset(&callbacks, 0, sizeof(callbacks));
callbacks.restart_device = app_commission_svr_restart_cb;
callbacks.save_startup = app_commission_svr_save_cb;
callbacks.restore_startup = app_commission_svr_restore_cb;
callbacks.reset_startup = app_commission_svr_reset_cb;

cluster = ZbZclCommissionServerAlloc(zb, ZCL_PROFILE_HOME_AUTOMATION,
    true /* Is APS Secured? */, &callbacks, arg);
```

4.3.1 InterPAN

InterPAN enable the commissioning server to operate on a selected channel and receive InterPAN messages from the commissioning tool without joining a network. The InterPAN messages are typically unsecured. Although it is possible to send APS secured InterPAN messages, it requires a special configuration. For a more common method of sending secured commissioning tool messages, see Section 4.3.2 which describes initially the joining the commissioning tool network.

```
struct ZbZclCommissionServerEnableInfoT info;

memset(&info, 0, sizeof(info));
info.page = 0;
info.channel = 11;
ZbZclCommissionServerEnable(cluster, &info);
```

The device is now ready to receive configuration and startup commands from a commissioning tool.

4.3.2 Join a commissioning tool network

The commissioning tool creates a temporary network for the device to be commissioned to join, this allows commissioning messages to be transferred directly without using InterPAN. The network should use the Zigbee® defined extended PAN ID: 00-50-C2-77-10-00-00-00, so the device easily finds it when performing a network discovery. By joining to the commissioning tool network it also eases the use of the APS security. The device joins the commissioning tool using a known pre-configured link key, allowing commissioning messages to be transferred using APS security, depending on what requirements has been set out by the application manufacturer.

4.4 Commissioning client

The commissioning client is used by the commissioning tool to perform the commissioning. The following code extract describes how to allocate and instantiate the ZCL commissioning client cluster.

```
struct ZbZclClusterT *cluster;
cluster = ZbZclCommissionClientAlloc(zb, ZCL_PROFILE_HOME_AUTOMATION,
    true /* Is APS Secured? */, &callbacks, arg);
```

4.4.1 InterPAN communication

Enables the commissioning client to operate on a selected channel and send InterPAN messages to the device being commissioned. The function is given in the following code extra

```
struct ZbZclCommissionClientEnableInfoT info;
memset(&info, 0, sizeof(info));
info.page = 0;
info.channel = 11;
ZbZclCommissionClientEnable(cluster, &info);
```

4.4.2 Forming a commissioning tool network

The commissioning tool forms a network for the devices being commissioned to join. This allows commissioning messages to be sent as unicast instead of as InterPAN.

The network must use the Zigbee® defined extended PAN ID of 00-50-C2-77-10-00-00-00 to allow the device to easily find the network while performing network discovery. Refer to the following code extract for an example.

```
struct ZbStartupT config;

/* Initialize the startup parameters */
ZbStartupConfigGetProDefaults(&config);

/* Form network with EPID = 0x0050c27710000000 */
config.startupControl = ZbStartTypeForm;
config.extendedPanId = ZB_EPID_GLOBAL_COMMISSIONING;

/* Configure the Commissioning Network preconfigured link key.
* In this example, for testing purposes, the ZigBeeAlliance09 key is used. */
memcpy(config.security.preconfiguredLinkKey, sec_key_ha, ZB_SEC_KEYSIZE);

/* Configure the channel list */
config.channelList.count = 1;
config.channelList.list[0].page = 0; /* Page 0 */
config.channelList.list[0].channelMask = 0x02108800; /* Channels: 11, 15, 20, 25 */

/* Call ZbStartup to form the network on one of the chosen channels. */
ZbStartup(zb, &config, callback, arg);
```

After the commissioning tool successfully forms the network, it enables the network permit join (for example `ZbNlmePermitJoinReq`).

The device to be commissioned then performs a `ZbStartup` with the `startupControl` set to `ZbStartTypeJoin` and the `extendedPanId` set to 00-50-C2-77-10-00-00-00, to ensure it only attempts to join the commissioning tool network.

4.4.3 Commissioning messages

Write attributes

To write commissioning parameters to the device being commissioned, the `ZbZclWriteReq()` API is used. The ZCL write commands sent using InterPAN or directly, as discussed earlier. Sample pseudo code is given below.

```
static void
write_rsp_cb(const struct ZbZclWriteRspT *writeResp, void *cb_arg)
{
    /* Handle the response. Check the return status. */
}

main()
{
    enum ZclStatusCodeT status;
    struct ZbZclWriteReqT req;
    uint8_t uint8_val;

    memset(&write_req, 0, sizeof(write_req));

    /* Destination */
    write_req.dst.mode = ZB_APSDE_ADDRMODE_EXT;
    write_req.dst.extAddr = DUT_EXT_ADDR;
    /* If using InterPAN, endpoint is set to ZB_ENDPOINT_INTERPAN.
     * Otherwise, set the endpoint to the remote Commissioning
     * Server endpoint Id. */
    write_req.dst.endpoint = ZB_ENDPOINT_INTERPAN;
    /* Attribute to write */
    uint8_val = ZbStartTypeJoin;
    write_req.count = 1;
    write_req.attr[0].attrId = ZCL_COMMISSION_SVR_ATTR_STARTUPCONTROL;
    write_req.attr[0].type = ZCL_DATATYPE_ENUMERATION_8BIT;
    write_req.attr[0].value = &uint8_val;
    write_req.attr[0].length = 1;

    /* Send command */
    status = ZbZclWriteReq(cluster, req, write_rsp_cb, NULL);
    if (status != ZCL_STATUS_SUCCESS) {
        /* We were not able to send this command (malformed?) */
}
```

Send the Commissioning Save, Restore and Reset commands

To send a commissioning save startup parameters command, which will tell the receiver to save the selected startup configuration, the application calls the `ZbZclCommissionClientSendRestoreStartup()` API. Sample pseudo code is given below.

```
static void
save_rsp_cb(struct ZbZclCommandRspT *rsp, void *arg)
{
    /* Handle the response. Check the return status. */
}

main()
{
    struct ZbZclClusterT *commiss_client; /* Commissioning Client cluster */
    uint64_t dst_eui; /* EUI of device to commission */
    uint8_t dst_ep; /* Destination endpoint, if not using InterPAN */
    struct ZbZclCommissionClientSaveStartup req; /* command configuration */
    enum ZclStatusCodeT status;

    memset(&req, 0, sizeof(req));
    req.index = 0; /* Index of attribute set to save */

    /* Send the command */
    status = ZbZclCommissionClientSendSaveStartup(commiss_client,
        dst_eui, dst_ep, save_rsp_cb, NULL);
    if (status != ZCL_STATUS_SUCCESS) {
        /* We were not able to send this command (malformed?) */
    }
}
```

The restore and reset commands are configured and sent in a similar way.

The restore command is sent using the `ZbZclCommissionClientSendRestoreStartup()` API. This command will restore the selected startup configuration that was saved earlier.

The reset command is sent using the `Z*bZclCommissionClientSendResetStartup()` API. This command will reset the selected or all startup configurations back to factory defaults.

Send the Commissioning Restart command

To send a commissioning restart startup parameter command, which tells the receiver to perform the actual Zigbee® startup procedure, the application calls the `ZbZclCommissionClientSendRestart()` API. Sample pseudo code is given below.

```
static void
restart_rsp_cb(struct ZbZclCommandRspT *rsp, void *arg)
{
    /* Handle the response. Check the return status. */
}

main()
{
    struct ZbZclClusterT *commiss_client; /* Commissioning Client cluster */
    uint64_t dst_eui; /* EUI of device to commission */
    uint8_t dst_ep; /* Destination endpoint, if not using InterPAN */
    struct ZbZclCommissionClientRestartDev req; /* command configuration */
    enum ZclStatusCodeT status;

    memset(&req, 0, sizeof(req));
    /* Configure the startup options. In this case, tell the receiver device
     * to startup immediately */
    req.options |= ZCL_COMMISS_RESTART_OPTS_IMMEDIATE;
    /* dst_ext is the Extended Address of the device being commissioned */
    /* rsp_callback is the function to be called upon reception of the
     * Commissioning Restart Response, or possible error. */
    ZbZclCommissionClientSendRestart(commiss_client, dst_ext, &req,
        restart_rsp_cb, NULL);
    if (status != ZCL_STATUS_SUCCESS) {
        /* We were not able to send this command (malformed?) */
    }
}
```

5 Pre-configured startup

A pre-configured startup is used to make a device rejoin a network as though the device has joined previously.

5.1 Configuration

5.1.1 PANID

```
config.panId = PANID_OF_NETWORK;
```

5.1.2 Short address

```
config.shortAddress = DESIRED_SHORT_ADDRESS;
```

5.1.3 Extended PANID

```
config.extendedPanId = EXTENDED_PANID_OF_NETWORK;
```

5.2 ZCL commissioning cluster APIs

5.2.1 Startup

Call the Zigbee® stack startup function. The callback function provided is called once the startup has completed whether it succeeded or failed.

```
ZbStartup(zb, &config, callback, arg);
```

6 Touchlink commissioning

Touchlink is a form of commissioning mechanism where nodes only join when devices are within close proximity of each other. The commissioning messages are initially sent through unencrypted InterPAN, and either forms or joins a distributed network for Zigbee® communication. Touchlink only supports distributed networking and does not support a centralized network.

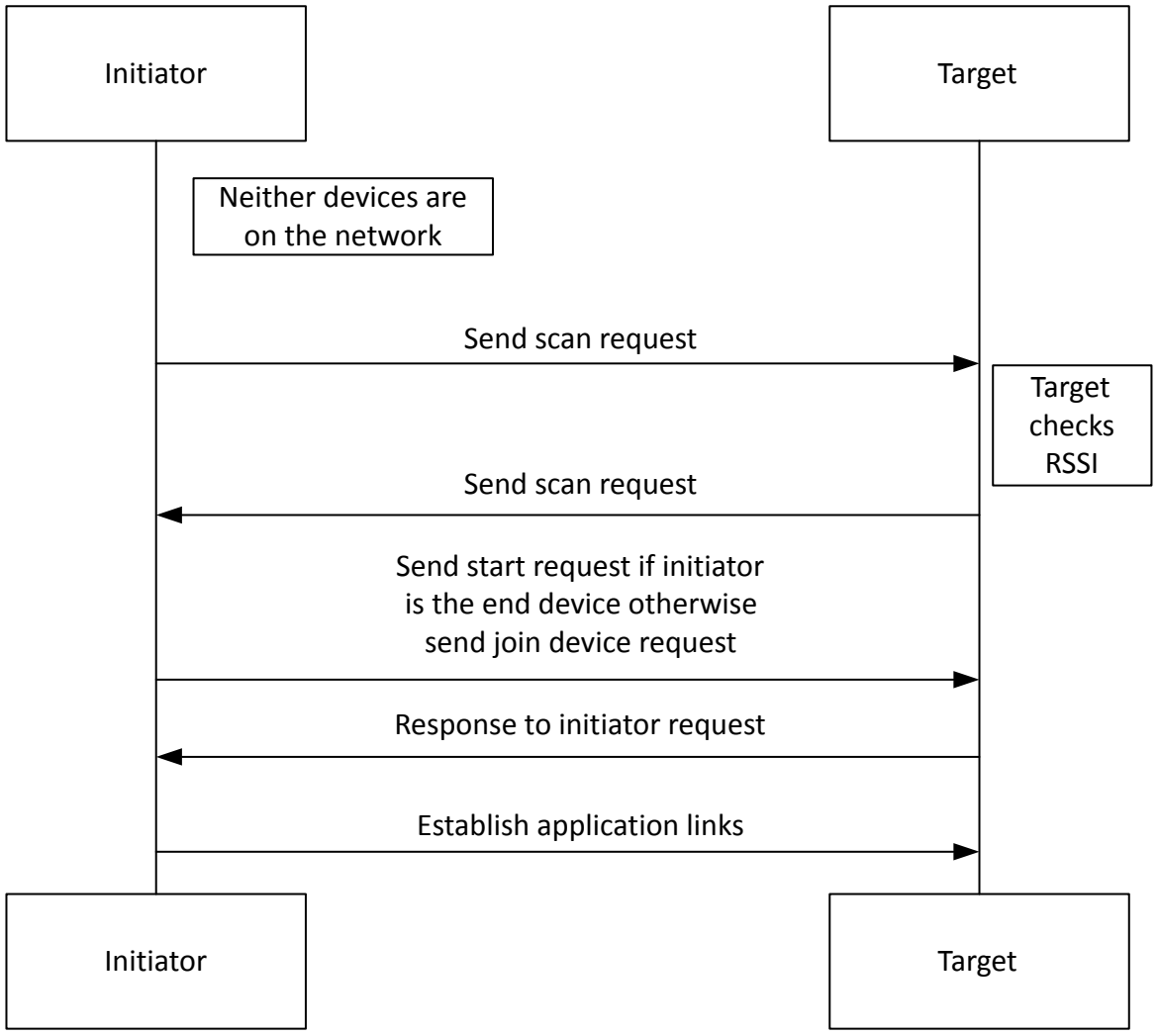
6.1 Flow chart

This section must be read in conjunction with Base Device Behavior Specification (13-0402) [2] - Sections 8.7 and 8.8.

6.1.1 Forming the network

To form a new Touchlink network, the devices must not be part of any other a network. The process is outlined in the figure below.

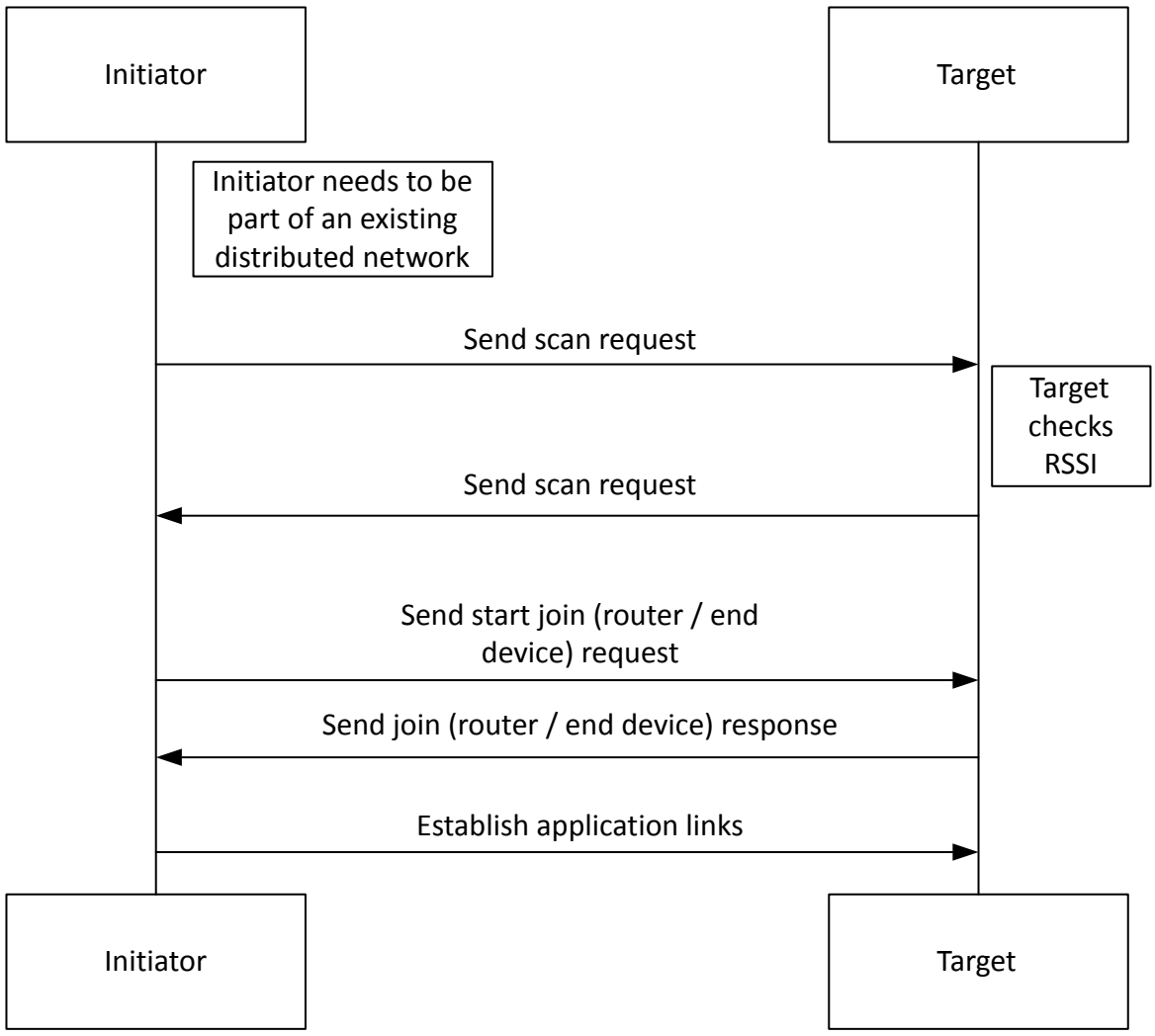
Figure 10. Forming the network



6.1.2 Joining subsequent devices

The figure below outlines the process for a device which joins an already existing network.

Figure 11. Joining subsequent device



Note: If the initiator is not on a network and is a router, it will look for a new distributed network. Any new additional devices joining must do so as a target.

6.2 Configuration

Configure the following `ZbStartupT` parameters when performing Touchlink.

6.2.1 BDB commissioning mode

```
/* Configure the Commissioning mode for Touchlink (0x1) */
config.bdbCommissioningMode |= BDB_COMMISSION_MODE_TOUCHLINK;
```

6.2.2 Configure Touchlink endpoint

```
/* Configure the endpoint for Touchlink Cluster */
config.touchlink.tl_endpoint = TOUCHLINK_ENDPOINT;
```

6.2.3 Configure application endpoint

```
/* Configure the endpoint for application bindings */
config.touchlink.bind_endpoint = APPLICATION_ENDPOINT;
```

6.2.4 Configure as Touchlink initiator

```
config.touchlink.flags = 0;
```

6.2.5 Configure as Touchlink target

```
config.touchlink.flags = ZCL_TL_FLAGS_IS_TARGET;
```

6.2.6 Configure as router

```
config.touchlink.zb_info = ZCL_TL_ZBINFO_TYPE_ROUTER;
config.touchlink.zb_info |= ZCL_TL_ZBINFO_RX_ON_IDLE;
```

6.2.7 Configure as end device

```
config.touchlink.zb_info |= ZCL_TL_ZBINFO_TYPE_END_DEVICE;
```

6.2.8 Changing Touchlink keys

```
/* Using the Uncertified Distributed Global Key
(d0:d1:d2:d3:d4:d5:d6:d7:d8:d9:da:db:dc:dd:de:df).
    * This key can be used as an APS Link Key between Touchlink devices. */
memcpy(config.security.distributedGlobalKey, sec_key_distrib_uncert,
ZB_SEC_KEYSIZE);

/* Use the Touchlink Certification Key
(c0:c1:c2:c3:c4:c5:c6:c7:c8:c9:ca:cb:cc:cd:ce:cf).
    * This key is "mashed" with the Touchlink session data to generate the Network
Key */
ZbBdbSet(zigbee_demo_info.zb, ZB_BDB_TLKey, sec_key_touchlink_cert,
ZB_SEC_KEYSIZE);
{
    /* Set the "Key Encryption Algorithm" to Certification */
    enum ZbBdbTouchlinkKeyIndexT keyIdx = TOUCHLINK_KEY_INDEX_CERTIFICATION;
    ZbBdbSet(zigbee_demo_info.zb, ZB_BDB_TLKeyIndex, &keyIdx, sizeof(keyIdx));
}
```

6.3 Touchlink commissioning APIs

6.3.1 Touchlink commissioning startup

Call the Zigbee® stack startup function. The callback function provided in the `ZbStartup()` function is called once the startup has completed whether it succeeded or failed.

```
ZbStartup(zb, &config, callback, arg);
```

6.3.2 Touchlink reset

The device must be set up as an initiator and the following flag must be set:

```
config.touchlink.flags |= ZCL_TL_FLAGS_FACTORY_RESET
```

Then call `ZbStartup` again.

6.3.3 Touchlink network update

```
zcl_touchlink_nwk_update_req(zb, extaddr, update_id)
```

7 Finding and Binding

Finding and Binding is the process of automatically establishing cluster bindings between matching input and output clusters. The cluster binding allows clusters to communicate with each other without the application needing to remember the target cluster address. Finding and Binding relies on the ZCL identify cluster. The ZCL identify cluster must be on the same endpoint as the clusters which needs binding.

7.1 Start the "Find and Bind" process automatically

To enable automatic Finding and Binding of clusters and endpoints during the joining procedure, the application must enable the BDB_COMMISSION_MODE_FIND_BIND bit in the `bdbCommissioningMode` parameter of the `struct ZbStartupT` startup attribute set.

When any device wants to join a network, the serve cluster device must first enable the Identify Server cluster to allow the Finding and Binding process to take place. The purpose of the Identify Server cluster is to start the device identification process.

Once the new device successfully joins the network, it must perform the following steps:

1. Allocate an Identify Client cluster on endpoint 0xFE (254) if it does not already exist.
2. Using the identify client, send a loop-back ZCL write request to any identified server clusters on the local device to start the identification process. This process is available for the next 10 seconds.
3. Search through the local endpoints for any client clusters. If none found, Finding and Binding is terminated.
4. From the identified client cluster, an identify query command is broadcast to the network. The responses are recorded including network address and source endpoint. If no responses are received, Finding and Binding is terminated.
5. Looping through the identify query responses, a ZDO simple descriptor request is sent to the remote device matching the network address and endpoint of the identified query responses. If the extended address of the remote device is not known, a ZDO IEEE address request is first sent to the remote device before sending the simple descriptor request. The extended address of the remote device is needed to perform any bindings between the devices.
6. The ZDO simple descriptor response is used to match up any clusters on the local device with the remote device that are to be bound.
7. There are three special clusters:
   - Alarm Client cluster
   - Messaging cluster
   - Poll control cluster

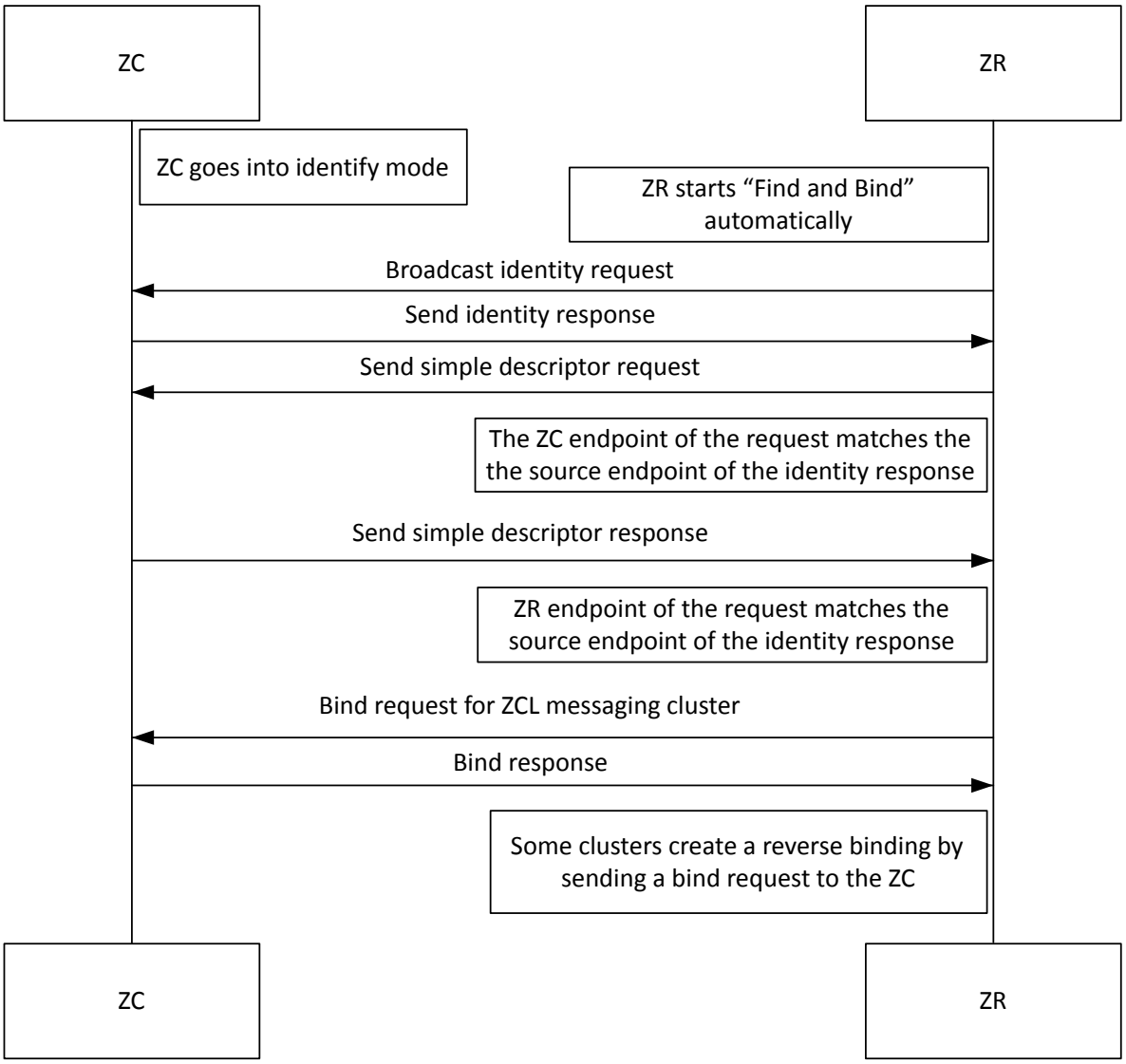
   The binding is done in the reverse direction, from server to client. If the local endpoint includes these clusters as clients, besides the normal binding, it will also send a ZDO bind request to the remote device to create a binding from server back to the client.
8. This process repeats with the next endpoint on the local device containing a client cluster.

7.2 Start Find and Bind manually

The Finding and Binding process can be initiated at any time while on a network. Any remote device with a server cluster the installer wants to allow Finding and Binding with, must first enable the Identify Server cluster to start the identification process on the devices. The local device application is then triggered to call `ZbStartupFindBindStart` to start the Find and Bind process. The process steps are the same steps as listed in Section 7.1 .

7.3 Finding and Binding flowchart

Figure 12. **Finding and Binding Chart for ZR**



Revision history

Table 3. **Document revision history**

Date	Version	Changes
02-Jul-2021	1	Initial release.

Contents

List of figures

List of tables