Application Note: JN-AN-1194b

ZigBee IoT Gateway - Host

This Application Note describes the hardware and software components that are required to implement the Host processor part of a Linux- based ZigBee Internet of Things (IoT) Gateway.

Creating a ZigBee IoT Gateway based on Linux allows the platform to utilise the vast array of software available for the operating system to create a highly featured, robust and scalable solution with a fast time to market.

Functions that are covered include:

- **ZigBee-to-JIP protocol translation**
- **Interaction with ZigBee devices in a WPAN**
- **IPv6 Route and Prefix advertisement**
- **IPv6 multicast routing**
- **Zeroconf based discovery of the ZigBee network**
- **Initial programming of the Border Router node**
- **IPv6 to IPv4 compatibility**
- **Compiling OpenWrt**

1 Application Overview

The Linux-based ZigBee IoT Gateway from NXP allows connection of a ZigBee network to the IPv6 internet via an application data transltion process where ZigBee devices can exchange data with any other IPv6 connected device, enabling the “Internet of Things”. For situations where a native IPv6 connection to the internet is not available, several transition mechanisms are available. This document describes the software on the host processor, which forms the “ZigBee IoT Gateway Host”. It must be interfaced with a “ZigBee IoT Gateway Control Bridge” described in the accompanying document JN-AN-1194 “ZigBee IoT Gateway – Control Bridge” in order to form a complete ZigBee IoT Gateway.

1.1 Hardware

The required hardware to create the system described in this Application Note is:

- Any hardware platform (PC, Embedded device) capable of booting a Linux operating system.
- JN5168 device to function as an IEEE802.15.4 WPAN interface (IoT Gateway ZigBee Control Bridge).

1.2 Operating System

The NXP IoT Gateway host functionality can be added to any Linux system, using common system packages and a few developed by NXP. The firmware used in the NXP IoT Gateway in the JN516x-EK001 Evaluation Kit is based upon the OpenWrt Linux distribution. This distribution was chosen because it has support for many cheap commercial off-the-shelf WiFi routers, and is also easy to port to new hardware platforms such as the NXP IoT Gateway reference design (JN-RD-6040). It is very configurable and has a wide variety of packages available for installation that provide a solution to almost any networking requirement.

2 Architecture

2.1 Protocol View

This diagram shows the translations between the various network protocols that are performed in the system. Client devices use the JIP protocol to access information on virtual JenNet-IP devices within the Host processor in the Gateway. The Host also runs the ZigBee-JIP daemon which converts accesses to the virtual devices into requests and commands to the ZigBee Control Bridge. The requests and commands are transported to the Control Bridge using a protocol over the bidirectional serial link between the Host and Control Bridge. The Control Bridge receives the requests and commands and translates them into ZigBee packets for sending to devices within the ZigBee wireless network. Information and responses coming from ZigBee devices within the network are similarly sent to the Host via the Control Bridge using the serial protocol

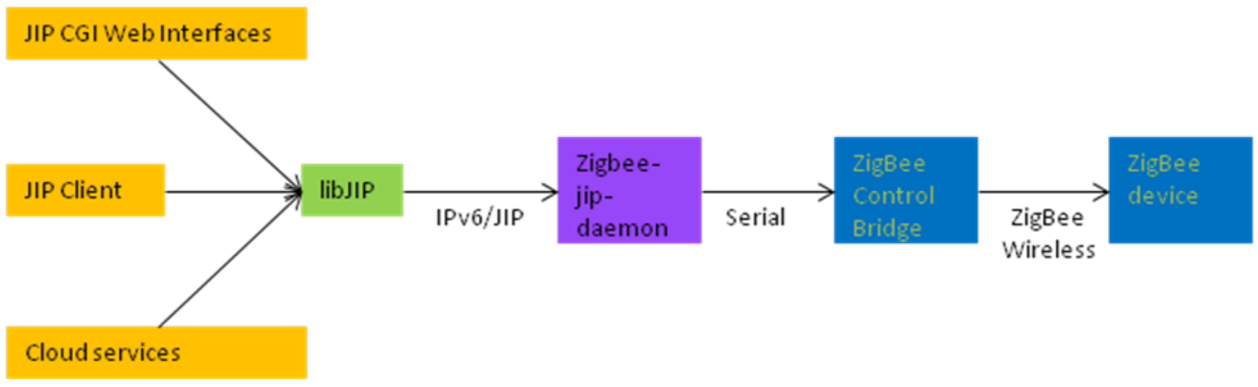

Figure 1: Protocols in the system

2.2 Network View

The diagram below shows the network layout. The gateway is connected to the LAN/WAN via the Ethernet port of the gateway, designated as the eth0 network device with IPv6 address **fd04:bd3:80e8:1::1**. For a definition of how IPv6 addresses are specified see http://en.wikipedia.org/wiki/IPv6_address. The gateway advertises a route to the IPv6 prefix **fd04:bd3:80e8:10::/64** which corresponds to the prefix of the virtual devices notionally within the ZigBee network. Incoming packets from the LAN/WAN that are destined for the ZigBee prefix (i.e. have the upper 64-bits of their destination address corresponding to this prefix) are routed across the gateway by the Linux kernel and arrive on a virtual network adaptor that has IPv6 addresses mapped to each node in the ZigBee network.

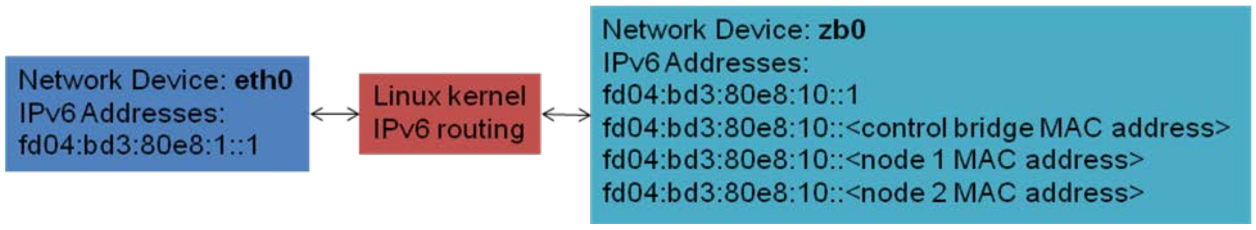

Figure 2: Device Addressing

3 Software Components

3.1 ZigBee / JIP interface (zigbee-jip-daemon)

The ZigBee / JIP daemon (**zigbee_jip_daemon**) creates a virtual network device in the gateway called **zb0** on which it registers IPv6 addresses for each node in the ZigBee network. It creates a socket for communications with the nodes, which is bound to the IPv6 addresses of the ZigBee nodes. The socket is used to host JIP MIBs for the devices (e.g. lights) on each node. Accesses to the node are passed to the ZigBee daemon which maps these into the required serial commands. These are then sent to the ZigBee Gateway Control Bridge node for sending into the network

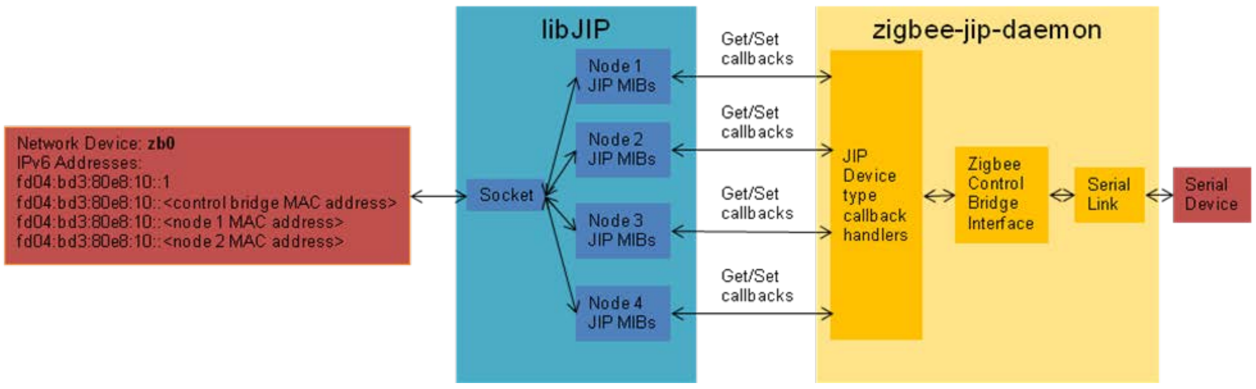

Figure 3: ZigBee / JIP Interfacing

3.1.1 Compiling zigbee-jip-daemon

Untar the **gzipped zigbee-jip-daemon** source archive. Change directory into the **JIP/Build** directory.

zigbee-jip-daemon depends on the following packages:

- **Avahi** (For Zeroconf discovery of the network)
- **LibJIP**

The dependency on **Avah**i can be removed by deleting the following line from the Makefile:

```
FEATURES ?= ZIGBEE-JIP_FEATURE_ZEROCONF
```

To compile **zigbee-jip-daemon**, run the makfilee in the **JIP/Build** directory.

3.1.2 Running zigbee-jip-daemon

zigbee-jip-daemon must be run as root in order to create the new network adaptor in the kernel using the tun driver. The only required parameter is the serial port to which the Control Bridge node is connected.

```
root@localhost # zigbee-jip-daemon –s /dev/ttyTX0
```

Many other parameters of the IEEE802.15.4 network can be set via command line options to **zigbee-jip-daemon**. These options can be listed by running the **zigbee-jip-daemon** command from a command line when logged into the gateway.

3.2 IPv6 Route and Prefix Advertisement (radvd)

IPv6 includes powerful automatic configuration services as part of the specification. The Linux IPv6 Router Advertisement Daemon (**radvd**) implements these parts of the specification and allows automatic configuration of routes and prefixes, amongst other details. Install it via the package manager of your distribution. Information regarding **radvd** may be found here: http://www.litech.org/radvd/.

Once **radvd** is installed, configure the following details in the configuration file (usually **/etc/radvd.conf**).

```
interface eth0
{
        AdvDefaultPreference low;
        AdvSendAdvert on;
        prefix fd04:bd3:80e8:1::/64
        {
                AdvRouterAddr on;
        };
        route fd04:bd3:80e8:10::/64
        {
                AdvRouteLifetime 3600;
                AdvRoutePreference medium;
        };
};
```

This configuration is for **eth0**, the Ethernet interface connecting the IoT Gateway to the rest of the network. The configuration tells **radvd** to:

- Set the preference of this router to low – this is set so that a default route to the IPv6 internet via another router should take precedence.
- Send periodic router advertisements and respond to router solicitations.
- Include the link local address of the router in the advertisements.
- Advertise a prefix of fd04:bd3:80e8:1::/64. This will be the prefix that machines connected to the Ethernet address will receive. If another IPv6 router exists on the network to allow machines to receive a global address, this is not necessary. The prefix advertisement includes the address of the router.
- Advertise a route to the prefix fd04:bd3:80e8:10::/64, which is the prefix used to map devices from the ZigBee network. Machines connected to the gateway's Ethernet interface will establish a route to the prefix automatically.

After setting this up and starting **radvd**, machines connected to the Ethernet interface should automatically acquire an IPv6 address with global scope and a route to the ZigBee network.

3.3 IPv6 Multicast Routing (mrd6)

IPv6 includes support for multicast to multiple hosts as an integral component of the specification. The ZigBee IoT Gateway makes use of incoming IPv6 multicasts to map to groups of devices within the ZigBee network. In order to route IPv6 multicasts between interfaces, a userspace multicast routing daemon is required, called **mrd6**. Install it via the package manager of your distribution. More information on **mrd6** may be found here: http://fivebits.net/proj/mrd6/.

IPv6 uses several ICMPv6 messages to implement the Multicast Listener Discovery (MLD) protocol. MLD allows IPv6 multicast routers to determine which multicast groups are present on hosts connected to their interfaces, such that incoming multicast packets may be forwarded as appropriate. The Control Bridge node in the gateway interprets IPv6 multicast addresses and converts them into ZigBee group addresses, and advertises these addresses to the gateway host where the **mrd6** daemon receives them. This allows **mrd6** to create destination mappings for each of these groups on the ZigBee network interface. If any IPv6 multicast packets destined for any of these groups are received on other interfaces, **mrd6** will create a source mapping and begin forwarding the multicast packets to the ZigBee interface. The IoT Gateway Control Bridge node will map these into ZigBee group communications and send them into the ZigBee network.

No special configuration of **mrd6** is required to function in this mode.

3.4 Interaction with the ZigBee JIP MIBs - libJIP

The JIP protocol is fully described in JN-UG-3080 and JN-UG-3086. NXP provides a library for interfacing with JenNet-IP devices or in this case ZigBee devices. From the JIP perspective, the ZigBee devices are virtual devices mapped to their JIP representations via the **zigbee-jip-daemon** running the JIP protocol from a Linux machine called **libJIP**. This may be linked to a custom application, or used via a command-line program. The API to this library for use in a custom application is described in JN-UG-3086.

3.4.1 Compiling libJIP

Untar the gzipped **libJIP** source archive. Change to the **Build** directory of the unzipped source tree.

libJIP depends on the following packages:

- **libxml2** – This is used to save and load network caches in **xml** form.

The dependency on **libxml2** can be removed by deleting the following line from the Makefile:

```
FEATURES ?= LIBJIP_FEATURE_PERSIST
```

However, this will render **libJIP** unable to preload network information and therefore all information must be discovered from the ZigBee network every time it is needed.

To compile the library, run the **Makefile** in the **Build** directory. The compiled library is output in the directory **../Library**.

libJIP also includes a command-line client for interacting with the network nodes which is output into the **Build** directory. The command-line client is documented in the JenNet-IP User Guide JN-UG-3086.

3.5 Interaction with the ZigBee JIP MIBs(JIP Web interfaces)

The NXP ZigBee IoT Gateway includes a set of web interfaces to interact with the network. There is a generic browser which allows all MIBs and variables on all nodes to be examined and updated, and also a more targeted interface (Smart Devices) which allows devices to be controlled by their functions, for example, dimming lights.

3.5.1 Compiling the JIP Web Interfaces

Extract the **JIPweb** tarball and change directory to the Build directory of the extracted source tree. The JIP web interface depends on the following packages:

- **Avahi** - for Zeroconf discovery of the network
- **libJIP** - for interacting with the network.

To compile the JIP web interfaces, run the **Makefile** in the **Build** directory.

3.5.2 Making changes to the JIP Web interfaces

The core of the web interface is the following cgi program:

- **/www/cgi-bin/JIP.cgi** – This implements a generic JenNet-IP control interface. It can be requested to take actions by specifying GET parameters. It returns results as JSON which can then be parsed by Javascript to update the web page. The source for this program is **Source/JIP_cgi.c**. The API to this program is described in Section 3.5.3.

There are also files within the NXP IoT Gateway's filesystem which may be edited either on, the gateway, or on a PC and then copied over to the NXP IoT Gateway:

- **/www/style.css** – This is the stylesheet for the JIP web interfaces. It may be modified to change the colours and some aspects of the layout of the page.
- **/www/NXP_Logo.gif** – This is the company logo that is located in the top right hand corner of the web interfaces. It may be changed to other images.

- **/www/js/JIP.js** – This is a Javascript library for issuing JIP requests to **JIP.cgi**. Its API is described in section 3.5.4
- **/www/Browser.html** – This is a jQuery and AJAX-powered version of a generic JenNet-IP browser. The page layout and style may all be edited in the HTML of the web page. The content is generated by Javascript making asynchronous calls to the **JIP.cgi** program.
- **/www/SmartDevices.html** – This is a jQuery and AJAX powered implementation of an interface for interacting with JenNet-IP smart light bulbs. It allows control of individual and groups of bulbs. The page layout and style may all be edited in the HTML of the web page. The content is generated by Javascript making asynchronous calls to the **JIP.cgi** program.

3.5.3 JIP.cgi API Reference

The **JIP.cgi** program may be run by making requests to the webserver on the NXP IoT Gateway. The parameters passed as part of the request determine the action taken by **JIP.cgi**. The results of the action are passed back to the web browser as JSON objects that may be interpreted by Javascript.

GET or POST Parameters

The following parameters may be passed as POST parameters to **JIP.cgi**

Parameter	Argument	Description
action		This argument informs **JIP.cgi** what request is being made of it. Each possible action is described below
	getVersion	Retrieve **JIP.cgi** and **libJIP** version information
	discoverBRs	Find list of JIP Border Routers available from this Gateway host
	discover	Perform JIP discovery of a network connected to a selected JIP Border Router
	GetVar	Read a variable on a remote node
	SetVar	Set a variable on a remote node
BRaddress	String representation of JIP Border Router IPv6 address	

The returned JSON object contains a “Status” object and data relevant to the JIP request. The status object contains number and string representations of the JIP status resulting from the request.

Example object:

```
        “Status”: { “Value”: 0, “Description”: “Success” }
```

Each supported action of **JIP.cgi** is described in the following sections.

3.5.3.1 Get Version Information (action=getVersion)

This action returns version information for the **JIP.cgi** program and **libJIP**. The returned object contains a “Version” object. This contains each components name and a version string.

Example object:

```
“Version”: { “JIPcgi”: “0.1 (r480000)”, “libJIP”: “0.15
(r480000)” }
```

3.5.3.2 Discover Control Bridge nodes (action=discoverBRs)

This action uses Zeroconf to perform a discovery of available gateway nodes at the webserver. The returned object contains a “BRList” object which is an array of the IPv6 addresses of discovered gateway nodes, represented as strings.

Example object:

```
“BRList”: [ “fd04:bd3:80e8:1:215:8d00:1234:5678” ]
```

3.5.3.3 Discover a Network using JIP (action=discover)

This action discovers the network of nodes connected to a gateway node. The IPv6 address of the JIP Border Router must be passed using the BRaddress parameter. The returned object contains a “Network” object representing the discovered network. The network object contains an array of nodes. Each node object contains a string representation of its IPv6 address, its Device ID as an integer, and an array containing its MIBs. Each MIB object contains its ID as an integer, its name as a string and an array containing its variables. Each variable object contains its name as a string, its index as an integer, its type as an integer, its access type as an integer and its security type as an integer.

Example object:

```
“Network”: { “Nodes” : [ {
“IPv6Address”=”fd04:bd3:80e8:1:215:8d00:1234:5678”, “DeviceID”:
2177865, “MiBs”: [ { “ID”: 50678885, “Name”: “Example MIB”,

“Vars”: [ { “Index”: 0, “Name”: “Int8Var”, “Type”: 0,
“AccessType”: 2, “Security”: 0 } ] } ] } ] }
```

3.5.3.4 Get the Value of a Variable (action=GetVar)

This action reads a variable on a node in the network. The following arguments must also be specified:

- BRaddress – IPv6 address of JIP border router
- nodeaddress – IPv6 address of the node
- mib – The name of the MIB containing the variable
- var – The name of the variable to read

The following arguments are optional:

- refresh – If set to “yes” then **JIPcgi** will rediscover the network before connecting to the node. If set to “no” then a cached copy of the network contents from the previous discovery will be used to look up the requested variable.

The returned object is as Section 3.5.3.3, but only a single node, MIB and variable are returned. The variable object includes an additional pair named “Value”, containing the current value of the variable.

3.5.3.5 Set the Value of a Variable (action=SetVar)

This action writes a variable on a node in the network. The following arguments must also be specified:

- BRaddress – IPv6 address of JIP border router
- nodeaddress – IPv6 address of the node
- mib – The name of the MIB containing the variable
- var – The name of the variable to read
- value – String representing the value to set

The following arguments are optional:

- refresh – If set to "yes" then **JIPcgi** will rediscover the network before connecting to the node. If set to "no" then a cached copy of the network contents from the previous discovery will be used to look up the requested variable.

The returned object contains only the "Status" object.

3.5.4 JIP.js API Reference

To simplify access to **JIP.cgi**, there is a Javascript library included on the NXP IoT Gateway which can be used to create web applications using JIP.

The Javascript file should be included using the following HTML tag:

```
<script src="js/JIP.js"></script>
```

The following is a description of the functions provided by this library.

3.5.4.1 Get Version Information: JIP_GetVersion(callback)

This function requests the version information from **JIP.cgi**, then calls the specified callback function with the received JSON object (see Section 3.5.3.1) as its parameter. The application should check the status of the request before attempting to use the version information.

3.5.4.2 Discover Border Router Nodes: JIP_DiscoverBRs(callback)

This function updates a list of available Border Router nodes within **JIP.js**. This list may be accessed using the name BRList. Once the list has been discovered, the callback function is invoked, passing the "Status" object of the request as its parameter. The application should check the status of the request before attempting to use the information in the BRList variable.

3.5.4.3 JIP_Context Object

A JIP_Context object is created to refer to the network behind a JIP Border Router. It is possible within **JIP,js** to create multiple contexts, each referring to a separate network. The JIP_Context provides a list of the available nodes via its Network attribute.

3.5.4.3.1 Connect to a Border Router using JIP_Context.Connect(IPv6Address)

This method may be used to associate the context with a given Border Router IPv6 address.

3.5.4.3.2 Discover a Network using JIP_Context.Discover(callback)

This method (re)discovers the network of nodes connected to a JIP Border Router previously associated with the context via the **Connect()** method. The function is passed a callback function to call with the results of the operation. Upon completion, the Network attribute of the JIP_Context is updated. The callback function is passed the “Status” object of the request, and a reference to the JIP_Context as its parameters. The application should check the status of the request before attempting to use the information in the JIP_Context.

3.5.4.3.3 Lookup Variables using JIP_Context.LookupVars(params)

This method returns a list of variable objects matching the constraints provided in the params object. Constraints are passed as members within this object and may include:

- **IPv6Address (string)**: Consider only the node with this IPv6 address
- **mibID (integer)**: Consider only the MIB with given ID
- **mibName (string)**: Consider only the MIB with given name
- **varIndex (integer)**: Consider only variables with the given index
- **varName (string)**: Consider only the variable with the given name

For example, this function may be used to retrieve a list of variable objects representing the variables with name “Mode” in all MIBs, on all nodes within the network using the call:

```
vars = JIP_Context.LookupVars({ varName: “Mode” });
```

3.5.4.3.4 Get the Value of a Variable using variable.GetVar(callback, user)

This method of the variable object reads the current value of the variable. The function should be passed a callback function and, if required, some user data. When the operation has completed, the callback function is invoked. It is passed the status of the request, the user data and a reference to the variable object (which should now contain the value in the Data attribute). The application should check the status of the request before using the value. The user data may be used to pass context data through the JIP library to the callback function.

3.5.4.3.5 Set the Value of a Variable using variable.SetVar(value, callback, user)

This method of the variable object sets the value of a variable. The function should be passed the required new value, a callback function and, if required, some user data. When the operation has completed, the callback function is called. It is passed the status of the request and a reference to the variable object. The user data may be used to pass context data through the JIP library to the callback function.

3.6 Zeroconf Discovery of ZigBee Network (Avahi)

Zeroconf provides an easy way for local network services to be automatically discovered and used with no manual configuration. **Avahi** provides a complete implementation of the standard and is used to advertise the IPv6 address of the IoT Gateway Control Bridge, so that other local network devices may connect to the ZigBee network. Install it via the package manager of your distribution. More information on **Avahi can** be found here: http://avahi.org.

zigbee-jip-daemon includes support for using **Avahi** as a client to advertise the IPv6 address of the Control Bridge node to the local network, using the service name “_jip._udp”.

3.7 Initial Programming of the ZigBee Node

It may be desirable to allow the Gateway host to program the Gateway node with an initial **ZigBeeNodeControlBridge** image, for example, during first bootup in the factory. This may be achieved using the program **JennicModuleProgrammer**, which is a C implementation of the protocol documented in the Boot Loader Operation application note, JN-AN-1003.

For instructions on how to run the **JennicModuleProgrammer** integrated in the IoT Gateway Host firmware, see the ZigBee IoT Gateway User Guide (included in this Application Note package).

3.7.1 Hardware Requirements

When the JN5168 chip boots, it checks if the MOSI line is held low – if this is the case it enters bootloader mode. In the IoT Gateway hardware the MOSI and Reset lines of the JN5168 are connected to GPIO lines of the Host processor which allows the Host to set the JN5168 device into bootloader mode, and also to reset the device to exit bootloader mode under the control of the **JennicModuleProgrammer**

3.7.2 Compiling JennicModuleProgrammer

Untar the gzipped **JennicModuleProgrammer** source archive. Change directory into the Build directory of the unzipped file tree and run “make”. The Makefile will compile the program. The compiled binary should then be installed to the target IoT Gateway system in “**/usr/bin**”.

3.7.3 Running JennicModuleProgrammer

The program is supplied with a script **JennicModuleProgram,sh** which controls the GPIO lines connected to MOSI and Reset to set the JN5168 device into bootloader mode. This program should be run when logged into the Gateway Host processor. See the ZigBee IoT Gateway User Guide in this Application Note package for details on how to log into the gateway.

Invoke the script by typing

```
JennicModuleProgram.sh <filename>
```

where <filename> is the binary file to be programmed into the JN5168. It then runs the **JennicModuleProgrammer** with the correct serial device and baud rate and after the device has been programmed it performs a verification of the contents of the memory against the contents of the binary file.

To view a summary of the options and flags which may be used with **JennicModuleProgrammer**, type

```
JennicModuleProgrammer –v
```

which will display a list of the available options.

3.8 JIP4 IPv4 Compatibility

JenNet-IP supports an application layer IPv4 compatibility mechanism allowing the control of JenNet-IP devices which only have an IPv6 address from an IPv4-only network. The protocol for the application level encapsulation is detailed in Appendix A.

To provide the application layer compatibility, a userspace daemon, **JIPd**, is required to listen on IPv4 sockets and forward requests to the IPv6 addresses of target nodes. It also provides a TCP socket on IPv4 for a client to connect to, rather than using UDP datagrams.

3.8.1 Compiling JIPd

The JIPv4 compatibility daemon **JIPd** advertises the JIPv4 service to the local network using Zeroconf. This feature is provided by **Avahi** and may be disabled by removing the line

```
FEATURES ?= JIPD_FEATURE_ZEROCONF
```

from the Makefile, if required.

Untar the gzipped **JIPd** source archive. Change directory into the Build directory of the unzipped file tree and run the Makefile . The Makefile will compile the daemon. This should be installed to the target Border-Router system in "**/usr/sbin**".

3.8.2 Running JIPd

No command line arguments are mandatory for **JIPd**. With no arguments, the daemon will bind to all IPv4 addresses and listen for incoming UDP and TCP connections on the normal JIP port, 1873. Requests to these sockets will be forwarded to the IPv6 addresses of nodes in the ZigBee network. The host running the daemon must have an IPv6 route to the prefix of the ZigBee network in order to forward these requests.

4 OpenWrt

The OpenWrt Linux distribution was chosen due to its large hardware support, small footprint, ease of use and large set of available packages. OpenWrt uses a heavily modified Buildroot environment to control the build process.

4.1 Compiling OpenWrt

4.1.1 Prerequisites

The following software packages will be required to build an OpenWrt image from the sources supplied with this Application Note:

- Linux-based PC
- Host toolchain (gcc)
- Make utility
- Subversion and git
- Ncurses/ncurses-devel

4.1.2 Setting up the Build Environment

First the build environment for your target platform must be set up using the tarball from the application source package.

Create a new directory in your home directory to work in. Change directory into the directory and extract the openWRT source package using the following command:

```
$ tar –xzf openWRT_<version>.tar.gz
```

Change directory to the target for which you want to build (in this case we are building for the Linksys WRT160NL – other targets include the LPC32xx family, as used in the NXP IoT Gateway RD6040) using the following command:

```
$ cd OpenWrt/targets/wrt160nl
```

Setup the build environment by running the target Makefile using the following command:

```
$ make
```

The Makefile will check out a known version of OpenWrt's source tree into a directory **openWRT/targets/<target>/backfire**. It will then install the OpenWrt feeds and copy the NXP modifications and a default configuration into the checked-out file tree. The NXP modifications to the OpenWrt tree are held in 4 locations:

- **openWRT/backfire** – Contains a target-agnostic overlay to the source tree that is rsync-ed over the top of the OpenWrt checkout.
- **openWRT/targets/<target>backfire-changes** – Contains a target-specific overlay to the source tree that is rsync-ed over the top of the OpenWrt checkout and the target-agnostic changes. This contains things like setting per-target configuration files or additional packages.
- **openWRT/targets/<target>/backfire-patches** – Contains any patch files necessary to be applied to the OpenWrt file tree.
- **openWRT/targets/<target>/config** – Contains the target-specific configuration file to be copied into the OpenWrt file tree.

4.1.3 Install JenNet-IP Modifications into OpenWrt

Once the OpenWrt source tree has been checked out and modified with the NXP IoT Gateway changes, the source for the additional packages must be copied into the build directory.

Change directory to the build directory using the following command:

```
$ cd backfire
```

Create a directory for storage of the downloaded source of OpenWrt using the following command:

```
$ mkdir dl
```

Alternatively, you may wish to create a symlink to a permanent source storage location using the following command:

```
$ ln –s <path to source storage> ./dl
```

Now the source tarballs for the NXP JenNet-IP Border Router may be copied into the new directory using the following command:

```
$ cp <path to JenNet-IP border router source>/*.tar.gz dl/
```

4.1.4 Build OpenWrt

Once the source tree has been prepared and the NXP IoT Gateway source tarballs have been copied, the OpenWrt build system can be used to compile an image for the target. If any modifications to the included packages are desired, the OpenWrt configuration menu may be used make the change using the following command:

```
$ make menuconfig
```

Now run the OpenWrt build process using the following command:

```
$ make
```

If extra verbosity is required to trace errors, add “V=99” to the make command. The build process may take several hours the first time.

When the build is complete, the output binary files will be found in the directory:

```
openWRT/targets/<target>/backfire/bin/<architecture>.
```

The root file-system for the target will be found in the directory:

```
openWRT/targets/<target>/backfire/build_dir/target-<ISA-
clibrary>/root-<architecture>/
```

Individual files may be copied to a target from this directory, or it could be mounted as an NFS root file-system for quick debugging.

Appendix A: IPv4 interface – JIPv4

In order to allow client devices within an IPv4 network to communicate with the virtual devices in the ZigBee Gateway Host which are assigned IPv6 addresses, a protocol is required which is suitable for use over IPv4 infrastructure. This protocol must contain enough information to be routed to a target virtual node in the IPv6 address space of the IoT Gateway when messages are sent from the client, and for information sent back to the client to be associated with the virtual node where it was generated.

The link between the client and IoT Gateway can use either UDP or TCP.

The IoT Gateway must either have a publicly-known IPv4 address or if the gateway is situated behind a SOHO router exposing a single static IPv4 address, some other mechanism to direct internet traffic intended for the gateway through the home network.

In order to provide the bridging function between IPv4 and IPv6, and also to make it easier to deal with messages from the client, three extra fields have been added to the front of the JIP packet. This format is referred to as JIPv4 to denote that it is essentially JIP but suitable for use over IPv4, using either Ethernet or Wifi.

This format will be used between clients and an IoT Gateway, where it will be terminated.

The format of the packet is as follows

Version	Length	IPv6 Addr	JIP packet
8-bits	16-bits (16+N)	128-bits	N bytes

The format of the JIP packet, commands and responses is described in the *JenNet-IP WPAN User Guide (JN-UG-3080).*

- The 8-bit Version field will allow any changes to the JIPv4 protocol to be identified. The current value is 0x01.
- The 16-bit Length field contains the length of the remainder of the packet, i.e. the number of bytes remaining to be read from the incoming packet to complete this transaction.
- The 128-bit IPv6 Address field contains the 128-bit IPv6 address of the target or source device in the virtual device.

To discover the (virtual) ZigBee devices available to be controlled, the IPv4 client (e.g. smartphone on Internet) addresses the IoT Gateway Host directly using an address formed of the IPv6 prefix for the virtual network in the most-significant 64-bits and 0 in the least-significant 64-bits of the JIPv4 packet. This allows the list of devices and their IPv6 addresses to be returned to the client.

At the Gateway, a process (**jipd**) listening on the JIP UDP or TCP port (1873) receives incoming JIPv4 packets and records the source IPv4 address and port number. It then removes the raw JIP payload and sends it to the target IPv6 address in the Host (**zigbee-jip-daemon**). Replies to commands or requests will be sent back to the recorded IPv4 address and port.

Revision History

Version	Notes
1.0	First release
1.1	Package updated with extra software components
2.0	Updated with new web-UIs
2.1	zigbee-jip-daemon upgraded to version 1.4, which has fixed the issue in which the presence of mono bulbs prevented the discovery of the network from the Gateway.

Important Notice

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com