Application Note: JN-AN-1110b

JenNet-IP Border-Router Host

This Application Note describes the hardware and software components that are required to implement a Linux-based Host side of a JenNet-IP Border-Router (the Node side is described in an accompanying document).

Creating a JenNet-IP Border-Router based on Linux allows utilisation of the vast array of software available for it to create a highly featured, robust and scalable solution with a fast time-to-market.

Functions that are covered include:

- **IPv6 packet routing**
- **Interaction with the JenNet-IP network nodes**
- **IPv6 route and prefix advertisement**
- **IPv6 multicast routing**
- **Node authentication (whitelisting) when security is enabled**
- **Zeroconf-based discovery of the JenNet-IP network**
- **Node firmware distribution**
- **Initial programming of the Border-Router Node**
- **JIP4 IPv4 compatibility**
- **JenNet-IP network traffic shaping**
- **Compiling OpenWrt**

1 Application Overview

The Linux-based JenNet-IP Border-Router allows the connection of one or more IEEE802.15.4 WPANs (Wireless Personal Area Networks) to the IPv6 Internet where they 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 described. This document describes the software on the host processor, which forms the “Border-Router Host” that connects to an IPv6 LAN/WAN. In order to form a complete Border-Router, it must be interfaced with a “Border-Router Node” that connects to the WPAN – here, the WPAN is a JenNet-IP network. The Border-Router Node is described in an accompanying document.

2 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 (Border-Router Host)
- JN5168 wireless microcontroller to function as an IEEE802.15.4 WPAN interface (Border-Router Node)

3 Operating System

The NXP JenNet-IP Border-Router functionality can be added to any Linux system, using common system packages and a few NXP developed ones. The NXP JenNet-IP Border-Router provided 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. It is very configurable and has a wide variety of packages available for installation that provide a solution to just about any networking requirement.

4 Software Components

4.1 IPv6 Packet Routing – 6LoWPANd

The most fundamental function of the Linux-based JenNet-IP Border-Router is to route IPv6 packets between the IEEE802.15.4 WPAN and the LAN/WAN. The daemon **6LoWPANd** (the 6LoWPAN routing Daemon) provides this function. Figure 1 below shows an overview of the architecture around **6LoWPANd**.

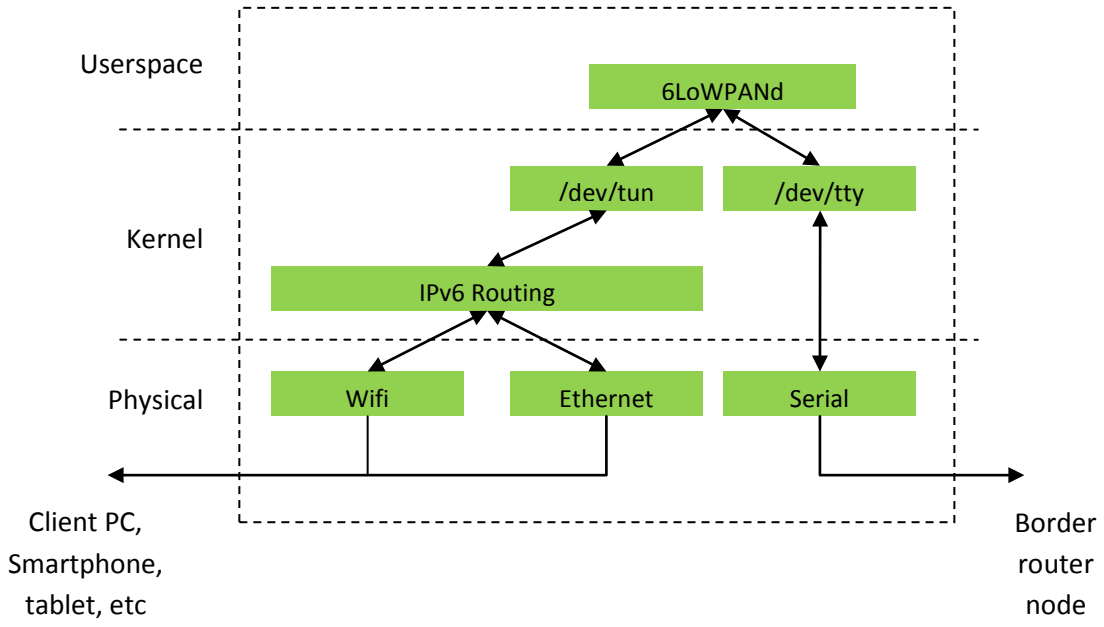


Figure 1: Role of 6LoWPANd

6LoWPANd uses the kernel "**tun/tap**" (http://en.wikipedia.org/wiki/TUN/TAP) virtual network interface driver to pass IP packets between the kernel and userspace. It is a standard kernel driver called "**tun.ko**".

The Border-Router Node is connected via a serial interface from UART0 on the JN5168 device to an available UART on the host processor. This can be via a USB-serial converter, if required.

The Border-Router Node (which is also the WPAN Co-ordinator) implements a simple serial protocol (see Appendix 1) that allows the configuration of network parameters on the Border-Router Node, and the passing of IPv6 packets between the node and the host. **6LoWPANd**

sends and receives IPv6 packets via the serial connection and passes these to and from the kernel via the **tun** driver.

4.1.1 Compiling 6LoWPANd

Untar the gzipped **6LoWPANd** source archive. Change to the **Build** directory.

6LoWPANd depends on the following packages:

- **Avahi** (for **Zeroconf** discovery of the network)

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

```
FEATURES ?= 6LOWPAND_FEATURE_ZEROCONF
```

To compile **6LoWPANd**, run the Makefile in the **Build** directory.

4.1.2 Running 6LoWPANd

6LoWPANd 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 Border-Router Node is connected.

```
root@localhost # 6LoWPANd –s /dev/ttyS0
```

Many other parameters of the IEEE802.15.4 network can be set via command line options to **6LoWPANd**.

Once **6LoWPANd** has started and is connected to the Border-Router Node, it will create a new network interface named **tun0** (this may be configured via the `–interface` parameter to **6LoWPANd**). This interface must be brought up and given a valid IPv6 address. A link local address is all that is required. A route to the IPv6 prefix of the JenNet-IP network may also be set up.

```
root@localhost # ifconfig tun0 up
root@localhost # ip -6 addr add fe80::1 dev tun0
root@localhost # ip -6 route add <JenNet-IP Prefix>/64 dev tun0
```

Once the IEEE802.15.4 network has been established, **6LoWPANd** queries the Border-Router Node for its IPv6 address. **6LoWPANd** saves this address in a text file named "**/tmp/6LoWPANd.<Interface name>**", as well as establishing an address record with **Avahi**, if **Avahi** support was compiled in. It should now be possible to ping the IPv6 address of the WPAN Co-ordinator.

```
root@localhost # ping6 `cat /tmp/6LoWPANd.tun0`
PING fd04:bd3:80e8:2:215:8d00:12:147d
(fd04:bd3:80e8:2:215:8d00:12:147d): 56 data bytes
64 bytes from fd04:bd3:80e8:2:215:8d00:12:147d: seq=0 ttl=255 time=11.082 ms
64 bytes from fd04:bd3:80e8:2:215:8d00:12:147d: seq=1 ttl=255 time=9.977 ms
64 bytes from fd04:bd3:80e8:2:215:8d00:12:147d: seq=2 ttl=255 time=9.808 ms
64 bytes from fd04:bd3:80e8:2:215:8d00:12:147d: seq=3 ttl=255 time=9.505 ms
```


Once the localhost can ping the IPv6 address of the Border-Router Node, the routing information for other machines on the LAN can be set up to allow them to also route to it. This can be set up manually as follows. First, determine the link local address of the Ethernet interface of the Border-Router via the command `ip -6 addr`:

```
root@localhost # ip -6 addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::224:a5ff:fed8:1240/64 scope link
       valid_lft forever preferred_lft forever
```

Once the link local address is known, a route to the IPv6 prefix of the JenNet-IP network can be set up on another machine via the following commands (Linux):

```
root@localhost # ip -6 route add <JenNet-IP Prefix>/64 via <Border router
Link Local Address> dev eth0
```

The machine must also be given an additional IPv6 address with global scope so that it can send packets off link to the JenNet-IP network using the following commands (Linux):

```
 root@localhost # ip -6 addr add fd04:80e8:bd3:1::2 dev eth0
```

Following this, the machine should also be able to ping the IPv6 address of the Border-Router Node.

4.2 IPv6 Route and Prefix Advertisement - radvd

IPv6 includes powerful automatic configuration 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:bd3:80e8:2::/64
        {
                AdvRouteLifetime 3600;
                AdvRoutePreference medium;
        };
};
```

This configuration is for eth0, the Ethernet interface connecting the Border-Router to the rest of the network. This 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 preference.
- 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:bd3:80e8:2::/64. Machines connected to the Border-Router'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 JenNet-IP network. They should be able to ping the IPv6 address of the Border-Router Node without any manual configuration.

4.3 IPv6 Multicast Routing – mrd6

IPv6 includes support for multicast to multiple hosts as an integral component of the specification. JenNet-IP makes use of IPv6 multicasts to implement group functionality. In order to route IPv6 multicasts between interfaces, a userspace 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, so that incoming multicast packets may be forwarded as appropriate. The nodes in a JenNet-IP network send MLD messages through the network to the Border-Router Node indicating their group membership. The Border-Router Node passes these on to the Border-Router Host where **mrd6** receives them. This allows **mrd6** to create destination mappings for each of these groups on the JenNet-IP network interface. If any IPv6 multicast packets are received on other interfaces which are destined for any of these groups, **mrd6** will create a source mapping and begin forwarding the multicast packets to the JenNet-IP interface. The Border-Router Node will then send these into the JenNet-IP network.

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

4.4 Interaction with the JenNet-IP Network Nodes - libJIP

The JenNet-IP (JIP) protocol is fully described in the NXP JenNet-IP User Guides with part numbers JN-UG-3080 and JN-UG-3086. NXP provide a library called **libJIP** for interfacing with JenNet-IP nodes (running the JIP protocol) from a Linux machine. This may be linked to a custom application, or used via a command line program. The Application Programming Interface (API) to this library for use in a custom application is described in the *JenNet-IP LAN/WAN Stack User Guide (JN-UG-3086)*.

4.4.1 Compiling libJIP

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

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 JenNet-IP 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 in the **Build** directory. The command line client is documented in the *JenNet-IP LAN/WAN Stack User Guide (JN-UG-3086)*.

4.5 Interaction with JenNet-IP Network Nodes – JIP Web Interfaces

The NXP JenNet-IP Border-Router includes a set of web interfaces to interact with the network. There is a generic browser which allows all MIBs and associated variables on all nodes to be examined and updated.

4.5.1 Compiling the JIP Web Interfaces

Extract the **JIPweb** tarball and change to the **Build** directory. 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.

4.5.2 Making Changes to the JIP Web Interfaces

The core of the web interfaces is a cgi programs, **/www/cgi-bin/JIP.cgi** which implements a generic JenNet-IP control interface. It can be requested to take actions by specifying GET or POST parameters. It returns results as JSON which can then be parsed by Javascript to update web based user interfaces. The source for this program is **Source/JIP_cgi.c**. The API to this program is described in Section 4.5.3

There are also files which may be edited on the NXP JenNet-IP Border-Router's file system, or on a PC and copied over to the NXP JenNet-IP Border-Router:

- **/www/style.css** : This is the style sheet 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, but the path to it is hardcoded in the JIP cgi programs (**Browser.cgi** and **SmartDevices.cgi**), so the name must not be changed.
- **/www/js/JIP.js** : This is a Javascript library for issuing JIP requests to **JIP.cgi**. Its API is described in Section 4.5.4
- **/www/Browser.html** : This is a jQuery and AJAX powered generic JenNet-IP browser. The page layout and style may 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 interface for interacting with JenNet-IP smart light bulbs. It allows control of individual and groups of bulbs. The page layout and style may be edited in the HTML of the web page. The content is generated by Javascript making asynchronous calls to the **JIP.cgi** program

4.5.3 JIP.cgi API Reference

The **JIP.cgi** program may be run by making requests to the web server on the JenNet-IP Border-Router. 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** which request is being made of it. Each possible action is described below.
	getVersion	Retrieve **JIP.cgi** and **libJIP** version information
	discoverBRs	Find list of Border-Router Nodes available from this Border-Router Host.
	discover	Perform JIP discovery of a network connected to a selected Border-Router Node. *Required*
	GetVar	Read a variable on a remote node. *Required*
	SetVar	Set a variable on a remote node. *Required*
BRaddress	String representation of Border-Router Node's 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.

4.5.3.1 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)" }
```

4.5.3.2 Discover Border-Router Nodes, action=discoverBRs

This action uses **Zeroconf** to perform a discovery of available Border-Router Nodes at the web server. The returned object contains a "BRList" object which is an array of the IPv6 addresses of discovered Border-Router Nodes, represented as strings. Example object:

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

4.5.3.3 Discover a Network using JIP, action=discover

This action discovers the network of nodes connected to a Border-Router Node. The IPv6 address of the Border-Router node 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 } ] } ] } ] }
```

4.5.3.4 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 Border-Router Node
- *nodeaddress* – IPv6 address of the node
- *mib* – Name of the MIB containing the variable
- *var* – Name of the variable to read

The following argument is optional:

- *refresh* – If set to "yes" then **JIP.cgi** 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.
- *stayawake* – If set to "yes" then **JIP.cgi** will request that sleeping end devices stay awake to receive a further request.

The returned object is as in Section 4.5.3.3, but only a single node, MIB and variable matching the request are returned. The variable object includes an additional pair named "Value" containing the current value of the variable.

4.5.3.5 Set the Value of a Variable, action=SetVar

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

- *BRaddress* – IPv6 address of Border-Router Node
- *nodeaddress* – IPv6 address of the node
- *mib* – Name of the MIB containing the variable
- *var* – Name of the variable to read
- *value* – String representing the value to set

The following argument is optional:

- *refresh* – If set to "yes" then **JIP.cgi** 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.
- *stayawake* – If set to "yes" then **JIP.cgi** will request that sleeping end devices stay awake to receive a further request.

The returned object contains only the "Status" object.

4.5.4 JIP.js API Reference

To simplify access to **JIP.cgi**, there is a Javascript library included on the NXP JenNet-IP Border-Router 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 are descriptions of the functions provided by this library.

4.5.4.1 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 4.5.3.1) as its parameter. The application should check the status of the request before attempting to use the version information.

4.5.4.2 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 JIP_BRList. Once the list has been discovered, the callback function is called, 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 JIP_BRList variable.

4.5.4.3 JIP_Context object

JIP.js uses an object orientated design, with an object used to represent each discovered network. This object is of type JIP_Context and may be created as follows:

```
var ctx = new JIP_Context();
```

The context object supports the following methods.

4.5.4.3.1 Connect to a border router node: Connect(br)

This function associates a border router IPv6 address with a context. The border router is passed as a string representation of the IPv6 address of the border router node.

4.5.4.3.2 Discover a network: Discover(callback)

This function discovers the network of nodes in the JenNet-IP network associated with a connected border router node. The function is passed a callback function, to call with the results of the operation. Upon completion, the contexts representation of the network is updated. The member variable is named “Network”. The callback function is passed the “Status” object of the request and the context as its parameters. The application should check the status of the request before attempting to use the information in the context variable.

4.5.4.3.3 Get the Value of a Variable: GetVar(variable, callback, user)

This function reads the value of a variable. The function should be passed a reference to the variable’s object within the context object, a callback function, if required, some user data and optionally a time interval if this request should be made regularly. When the operation has completed, the callback function is called. It is passed the status of the request, the user data and a reference to the variable’s object within the context object. 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.

4.5.4.3.4 Set the Value of a Variable: SetVar(variable, value, address, callback, user)

This function sets the value of a variable. The function should be passed a reference to the variable’s object within the context object, the new value, an optional multicast IPV6 address, 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 the user data. The user data may be used to pass context data through the JIP library to the callback function.

4.6 Node Authentication - radiusd

When IEEE802.15.4 security is enabled in a JenNet-IP WPAN, joining nodes must be authenticated before the network encryption key is shared with them so that they may join the network. Node authentication is handled by the industry standard RADIUS protocol. The Border-Router Node includes a RADIUS client that performs the Password Authentication Protocol (PAP). The Border-Router Node will connect to the IPv6 address of a configured RADIUS server and attempt to authenticate every node that attempts to join the network.

To provide the RADIUS authentication server, the **freeradius2** package is used. Install it via the package manager of your distribution. More information on **freeradius2** may be found here: http://freeradius.org/.

When a node attempts to join the JenNet-IP network, it does so using a “commissioning” encryption key, unique to that node, as described in the *JenNet-IP WPAN Stack User Guide (JN-UG-3080)*. This culminates in the Border-Router Node sending a RADIUS “access request” packet to the configured RADIUS server. The server should respond with either an “access deny”, or an “access accept” message. The “access accept” message includes the commissioning key of the node in its payload.

The username and password that are presented to the RADIUS server are both a string representation of the MAC address of the node attempting to join. The shared secret

between the Border-Router Node and the RADIUS server is a string representation of the network encryption key.

The following configuration items need to be specified:

- Authorisation listen address: IPv6 address - for example, all: `ipv6addr = ::`
- Client entry for the Border-Router Node, using the network encryption key as its shared secret. This is added via a configuration directive such as:
  `client <Border router IPv6 address> { secret = <Network Key> }`
- List of allowed MAC addresses (usernames). It is suggested that this is included from another configuration file via a directive such as:
  `$INCLUDE <Path to JenNet-IP users file>`

Each user record has the following format:

```
<MAC Address of Node> Cleartext-Password := “<MAC Address of Node>”
  Vendor-Specific = “0x00006DE96412<Node Commisioning Key>”
```

The `Vendor-Specific` attribute is the vendor type 100 (0x64) of NXP Semiconductors (0x006DE9), which is 16 bytes long (0x12). The following is an example entry for a node with MAC address 00:15:8D:00:00:09:FD:C7 and commissioning key 0x11223344556677889AABBCCDDEEFF.

```
00158D000009FDC7 Cleartext-Password := “00158D000009FDC7
  Vendor-Specific = “0x00006DE964120011223344556677889AABBCCDDEEFF”
```

4.6.1 Node Greylisting

In most use cases, JenNet-IP systems will need a semi-automatic way of allowing nodes to join a network, once a user has selected nodes that are known to them. In the Border-Router supplied in NXP JN516x evaluation kits, this process is achieved using greylisting. NXP provide a patch to **freeradius2** that implements a new authentication module, called **rlm_greylist**. This patch may be found in the OpenWrt source tarball under the path:

```
OpenWrt/backfire/feeds/packages/net/freeradius2/200-rlm-greylist.patch
```

This new authentication module will claim authentication of users that have not been claimed by any other module, i.e. if the username cannot be found by any other module, this module will claim to have found it and handle the authentication. In handling the authentication, **rlm_greylist** will run a configured external script with the arguments “—user=<Username>”. It is then the responsibility of this script to perform any actions necessary to add the new node into a user interface for display and selection. This user interface should ask the user to confirm the MAC address of the newly joining node and its commissioning key - for example, by reading them from the packaging. If they confirm the MAC address and commissioning key then an entry for the node, with its commissioning key, can be created in the users’ file, allowing the node access to the network. If not, then the MAC address may be forgotten.

4.7 Zeroconf Discovery of JenNet-IP Network (Avahi)

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

6LoWPANd includes support for using **Avahi** as a client to advertise the IPv6 address of the Border-Router Node to the local network, using the service name "**_jip._udp**".

4.8 Node Firmware Distribution (OND)

JenNet-IP supports Over-the-Network Download (OND) of new firmware images to nodes. This functionality is provided by the Border-Router Host via a daemon called **FWDistributiond**. The daemon answers requests for firmware from nodes in the network, and will push new firmware into the network if instructed to do so. There is a command line client to the daemon provided in the source.

4.8.1 Compiling Firmware Distribution Tools

The firmware distribution daemon uses the kernels INOTIFY mechanism to monitor for firmware images in a directory. To use this feature, the kernel configuration option CONFIG_INOTIFY_USER should be selected.

Untar the gzipped **FWDistribution** source archive. Change to the **Build** directory and run the makefile, which should compile the daemon and command line client. These should be installed into the target Border-Router system in "**/usr/sbin**" and "**/usr/bin**" respectively.

4.8.2 Running the Firmware Distribution Daemon

The only parameters required by the firmware distribution daemon are either a list of firmware files that should be available for download (each specified via a "-f" command line option), or a directory that should be monitored for firmware images (if INOTIFY support is enabled), which is specified via a "-d" command line option. For example:

```
root@localhost # FWDistributiond –d /var/nxp/firmware/
```

The daemon should start and load in any compatible firmware files from the directory. It will then monitor this directory for further files being created or moved here. If they are valid firmware images then they will be loaded and made ready for distribution.

4.8.3 Running the Firmware Distribution Command Line Client

The built-in firmware distribution client can be used to control any aspect of the firmware distribution daemon. Firstly, it can be used to list the firmware images that are available using the "-l" command line option.

```
root@localhost # FWDistributionControl -l
List of available firmwares:
Device ID   Chip    Revision   Filename
0x801500e1  0x141c  491       0x3a773a77s_CH18_DeviceBulb_DR1050_JN5148_v491.ond
```

This list contains the files from the monitored directory (or passed as individual files when the daemon was started) that have been deemed to be a valid firmware image.

The daemon may be commanded to begin a broadcast download to the JenNet-IP network using the "-d" command line option. In this case, the device ID, chip type, revision and the IPv6 address of the Border-Router Node must also be specified, as follows:

```
# FWDistributionControl –d –I <DeviceID> -T <Chip Type> -R <Revision> -6
<IPv6 Address of Border router node> --block-interval=<time between
broadcast blocks>
```

The following parameters are specified

- `-d` : Initiate a download
- `-I` : Download an image appropriate for devices with the specified ID, if one is available
- `-T` : Download an image appropriate for devices of the specified chip type, if one is available
- `-R` : Download an image with the specified revision, if one is available
- `-6` : Download the image via the Border-Router Node specified
- `--block-interval` : In units of 10ms, the time to wait between broadcasts of each block in the image. The default is 100 (or 1 second) between blocks. This is suitable for smaller networks. In larger networks, or those carrying more traffic, this should be set to a larger value to slow down the broadcast. This makes more bandwidth available to other network services.

For example, the firmware listed previously may be downloaded from the Border-Router to the connected JenNet-IP network using this command:

```
root@localhost # FWDistributionControl -d -I 0x801500e1 -T 0x141c -R 491 -6
`cat /tmp/6LoWPANd.tun0`
```

The progress of a broadcast download may be monitored using the “-m” command line argument, as follows:

```
# FWDistributionControl –m –I <DeviceID> -T <Chip Type> -R <Revision> -6
<IPv6 Address of Coordinator>
```

For example, to monitor the progress of the broadcast download started previously, the following command may be run:

```
root@localhost # FWDistributionControl -m -I 0x801500e1 -T 0x141c -R 491 -6
`cat /tmp/6LoWPANd.tun0`
Monitoring status
fd04:bd3:80e8:2:215:8d00:12:147d    0xffffffffffffffff  0x801500e1  0x141c
491 -   285 /  3215 : 0
```

This shows the number of blocks of firmware that have been sent and the total size of the firmware image.

Once a complete image has been downloaded, the nodes must be commanded to reset and move to the new firmware image. This can be achieved using the “-r” command line argument, as follows:

```
# FWDistributionControl –r –I <DeviceID> -T <Chip Type> -R <Revision> -6
<IPv6 Address of Coordinator> --reset-timeout=<time to reset> --reset-
depth-influence=<time offset based on network depth> --reset-repeat-
count=<number of reset repeats> --reset-repeat-time=<time between reset
messages>
```


There are a few extra parameters to control how the reset occurs:

- `--reset-timeout` : In units of 10ms, how long to wait to reset after receiving the command
- `--reset-depth-influence` : This parameter can be used to make nodes that are lower in the JenNet tree wait longer before resetting than those higher up. This should cause the tree to collapse from the top to the bottom and then reform with minimal traffic. This parameter sets the number of milliseconds to add to the reset timeout per layer of depth in the tree
- `--reset-repeat-count` : By default, the reset command is broadcast into the network once, but this parameter can be used to specify that the reset command is to be sent multiple times
- `--reset-repeat-time` : By default, if multiple reset commands are sent, there will be a 200ms interval between consecutive commands. This parameter allows the user to increase or decrease this time. Each time a reset command is sent, the timeout (time until reset) within the command is decremented by this interval, so that the final reset time remains the same when multiple commands are sent

For example, to reset the nodes (that received the previously distributed firmware) after 6 seconds, the following command may be used:

```
root@localhost # FWDistributionControl -r -I 0x12345678 -T 0x0001 -R 2
-6 `cat /tmp/6LoWPANd.tun0` --reset-timeout=600
Requesting Reset of Device ID 0x12345678 via coordinator
fd04:bd3:80e8:2:215:8d00:12:147d in 6.00 seconds, depth influence=10
```

These nodes should now reset and begin running the new firmware image.

4.8.4 Creating a Custom Firmware Distribution Client

The firmware distribution daemon is controlled via a Unix socket that is used to pass messages between clients and the daemon. The protocol for this communication is in the **FWDistribution** source, in the file **Source/Common/FWDistribution_IPC.h**. The command line client (in **Source/Clients/CLI_main.c**) should be studied as an example.

4.9 Initial Programming of the Border-Router Node

The recommended way to upgrade the firmware in the Border-Router Node is to use OND. However, it may be desirable to allow the Border-Router Host to program the Border-Router Node with an initial JenNet-IP image - for example, during first boot-up in the factory. This may be achieved using the program **JennicModuleProgrammer**, which is a C implementation of the protocol documented in the Application Note *"Boot Loader Operation" (JN-AN-1003).*

4.9.1 Hardware Requirements

When the JN516x chip boots, it checks whether the MOSI line is held low – if so, it enters bootloader mode. It is therefore recommended that the host processor has some means to assert this and the reset line, so that it can automatically put the JN516x device into bootloader mode. This could be achieved by connecting these lines to the GPIO of the host processor.

4.9.2 Compiling JennicModuleProgrammer

Untar the gzipped **JIPd** source archive. Change to the **Build** directory and run the makefile, which should compile the program. The resulting file should be installed into the target Border-Router system in "**/usr/bin**".

4.9.3 Running JennicModuleProgrammer

First put the JN516x device into bootloader mode using GPIO, if connected, or push-buttons. When the device enters bootloader mode, it listens for commands at 1Mbaud for 100ms before dropping back to 38400 baud. So if **JennicModuleProgrammer** can be started within 100ms of the program/reset lines being asserted then "–initialbaud" and "–programbaud" should both be set to 1000000. Otherwise, set "-initialbaud" to 38400(default) and "–programbaud" to 1000000(default). Use "–serial" to specify the serial device to which the Border-Router Node is connected. The connected device should be identified as the right device type.

Once it can be verified that the device is communicating, specify the "–firmware" option with a binary file to program. The "–verify" option may also be specified to verify that the write to Flash memory has been successful.

4.10 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.

To provide the application layer compatibility, the 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.

4.10.1 Compiling JIPd

The JIPv4 compatibility daemon **JIPd** advertises the JIPv4 service to the local network using **Zeroconf**. This feature is provided by **Avahi** and, if not required, can be disabled by removing the line "FEATURES ?= JIPD_FEATURE_ZEROCONF" from the Makefile.

Untar the gzipped **JIPd** source archive. Change to the **Build** directory and run the makefile, which should compile the daemon. The resulting file should be installed into the target Border-Router system in "**/usr/sbin**".

4.10.2 Running JIPd

No command line arguments are mandatory to **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 JenNet-IP network. The host running the daemon must have an IPv6 route to the prefix of the JenNet-IP network in order to forward these requests.

4.11 JenNet-IP Traffic Shaping

The standard Linux Quality of Service (QoS) mechanisms may be used to shape the traffic through the interface to the JenNet-IP network. This may be desirable to prevent excessive traffic on the JenNet-IP network causing network overload. If this is desired, it is recommended to create a high-speed traffic class for traffic to the Border-Router Node, limited by the speed of the serial connection to the Border-Router Node. For the rest of the IPv6 prefix of the JenNet-IP network, a maximum data-rate of 5kbps is recommended.

The “tc” package and related kernel modules are required for this. Install them via your package manager.

The following example commands limit traffic to 900kbps to the Border-Router Node and 5kbps to the rest of the JenNet-IP network.

```
root@localhost # tc qdisc add dev <JenNet-IP net device> root handle 1: htb
default 12
root@localhost # tc class add dev <JenNet-IP net device> parent 1: classid
1:10 htb rate 900kbit ceil 900kbit
root@localhost # tc filter add dev <JenNet-IP net device> protocol ipv6
parent 1: u32 match ip6 dst <IPv6 address of Border Router Node> classid
1:10
root@localhost # tc class add dev <JenNet-IP net device> parent 1: classid
1:12 htb rate 5kbit ceil 5kbit burst 1 cburst 1
root@localhost # tc qdisc add dev <JenNet-IP net device> protocol ipv6
parent 1:12 handle 20: pfifo limit 5
```

5 Network Security and Firewall Guidelines

5.1 IEEE802.15.4 Security

The IEEE802.15.4 network is secured using 128-bit AES encryption. The network uses a single key shared amongst all authenticated devices. In order to join a network, a device must be authenticated. During this period, the device should use a unique commissioning key. This key must be shared with the Border-Router node such that it can distribute this key and allow the joining device to join. For maximum security, the commissioning key should be:

- Unique to each device
- Not derived from any information that is available ‘on air’
- Shared with the Border-Router via an out-of-band method that cannot be sniffed - for example, via NFC or QR code or printed text on the device. Of course, the target environment of the device must be considered for this selection and whether potential attackers will have physical access to the device - for example, if it may be placed outside a user’s home

For ease of demonstration, by default, this Application Note and the *JN-AN-1162 “JenNet-IP Smart Home”* Application Note use a commissioning key that is derived from the MAC address. This approach is not recommended for use in customer products because it leaves open a security hole allowing a malicious device to retrieve the network key in use.

5.2 JIP Security

JenNet-IP uses a protocol called JIP, built upon the industry standard protocol UDP. This protocol provides control and monitoring of devices via end-to-end IPv6 packets. Within the IEEE802.15.4 WPAN, these are AES128-encrypted at the network level, but JIP/UDP does not provide any form of authentication or authorization of commands. As such, allowing JIP traffic to flow between the JenNet-IP network and the public Internet is not recommended due to the potential for malicious parties to intercept packets from and send packets to JIP devices. NXP recommends the use of a gateway between the IEEE802.15.4 WPAN and the public Internet to ensure that only trusted devices may send packets into the IEEE802.15.4 WPAN and that packets leaving the IEEE802.15.4 WPAN are suitably secured using a protocol of the customer’s choice for transport on the public Internet.

5.3 IP Network Security (Firewall)

In order to prevent malicious JIP traffic reaching the JenNet-IP network, this Application Note software enables the IP firewall present in openWrt on Router-type devices such as the Linksys wrt160nl (Gateway-type devices, such as the NXP JN-RD-6040 IoT Gateway, do not use the firewall as it is assumed that the firewall protecting the Ethernet network of the IoT gateway will provide the required protection). This is the recommended approach for customers creating gateway devices that are connected to the public Internet. Only ports related to required services should be opened to the Internet, to limit the attack surface of the gateway and reduce the risk of malicious parties compromising the JenNet-IP network.

Linux provides a very powerful IP firewall which is configurable via the `iptables` and `ip6tables` commands. There is a wealth of information on the proper configuration of Linux iptables firewalls on the Internet - the project homepage is http://www.netfilter.org/projects/iptables/.

The following are guidelines on configuring a secure firewall using iptables:

- Set the INPUT and FORWARD policies to REJECT - this will drop any packets that do not match a specific rule
- REJECT all traffic from the public Internet going to the JIP default port (UDP 1873)
- ACCEPT traffic to ports that are specifically required by necessary services on the gateway
- Do not allow ssh access (TCP port 22) from the public Internet unless absolutely necessary

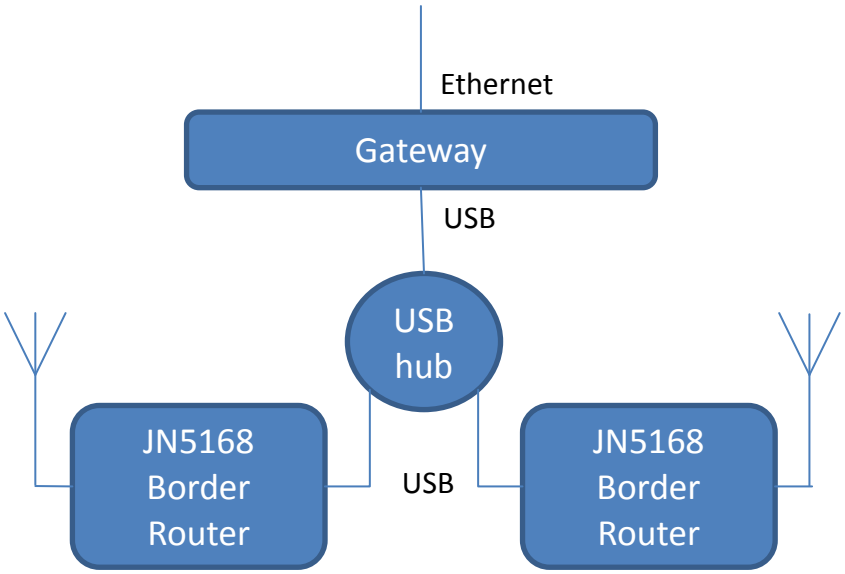
These rules must be applied separately for IPv4 and IPv6 using the `iptables` and `ip6tables` commands respectively.

6 Using Multiple Gateways

The JenNet-IP IEEE802.15.4 stack supports a maximum of 250 nodes in a single network. However, networks with a larger number of nodes than this may be formed through the use of multiple Border-Routers, each coordinating a network of up to 250 devices. Each IEEE802.15.4 WPAN has a separate IPv6 prefix but IP routing means that every device may be contacted by any other device, and all devices are similarly routable from the LAN/WAN.

6.1 Single Gateway with Multiple Border Routers

In this case, a single gateway device supports multiple connected Border-Router nodes forming multiple networks. This may be achieved, for example, by adding additional USB-serially connected Border-Router nodes via a USB hub.



In this example, each Border-Router is configured via 6LoWPANd to be on a separate /64 IPv6 prefix. Radvd is then used to advertise a route to both of these prefixes to the Ethernet network. The following are the openWrt configuration files necessary to set this configuration.

/etc/config/6LoWPAN

```
config 6LoWPANd 'tun0'
    option ignore '0'
    option tty '/dev/ttyUSB0'
    option baudrate '1000000'
    option interface 'tun0'
    option activityled 'none'
    option channel '11'
    option pan '0x1234'
    option jennet '0x11111111'
    option prefix 'fd04:bd3:80e8:2::'
    option qos_enable '0'
    option secure '1'
    option jennet_key '::1'
    option auth_scheme '1'
    option radius_ip 'fd04:bd3:80e8:1::1'
```

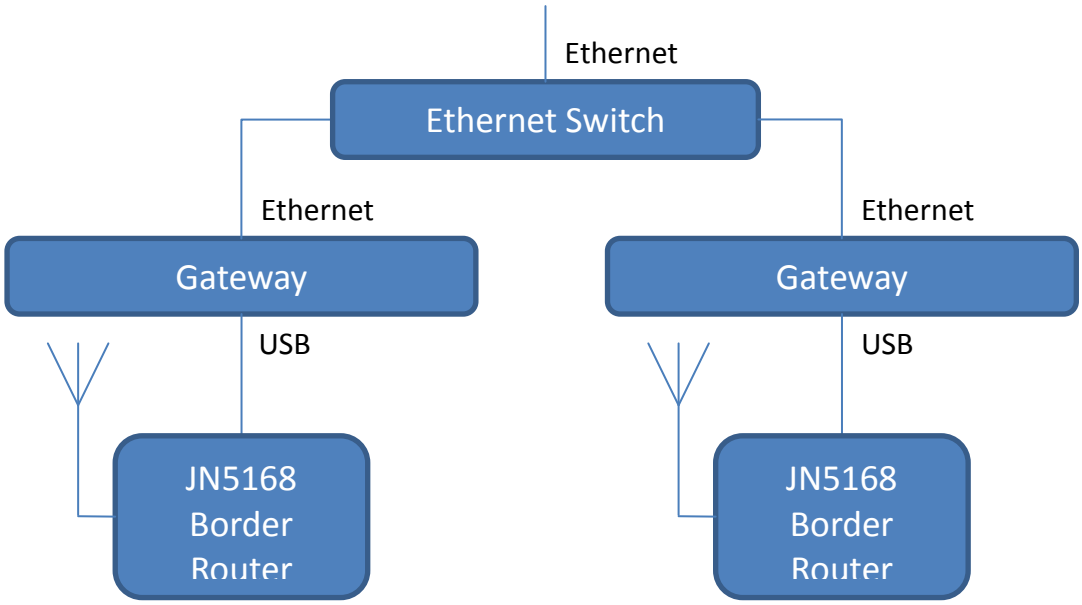
```
config 6LoWPANd 'tun1'
    option ignore '0'
    option tty '/dev/ttyUSB1'
    option baudrate '1000000'
    option interface 'tun1'
    option activityled 'none'
    option channel '11'
    option pan '0x1235'
    option jennet '0x11111111'
    option prefix 'fd04:bd3:80e8:3::'
    option qos_enable '0'
    option secure '1'
    option jennet_key '::1'
    option auth_scheme '1'
    option radius_ip 'fd04:bd3:80e8:1::1'
```

/etc/config/radvd

```
config 'interface'
        option 'interface' 'lan'
        option 'AdvSendAdvert' '1'
        option 'ignore' '0'
        option 'IgnoreIfMissing' '1'
        option 'AdvSourceLLAddress' '1'
        option 'AdvDefaultPreference' 'medium'

config 'prefix'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvOnLink' '1'
        option 'AdvAutonomous' '1'
        list 'prefix' 'fd04:bd3:80e8:1::/64'
        option 'AdvRouterAddr' '1'

config 'route'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvRouteLifetime' '3600'
        option 'AdvRoutePreference' 'medium'
        list 'prefix' 'fd04:bd3:80e8:2::/64'

config 'route'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvRouteLifetime' '3600'
        option 'AdvRoutePreference' 'medium'
        list 'prefix' 'fd04:bd3:80e8:3::/64'
```

6.2 Multiple Gateways with Individual Border-Routers

In this case, a single Border-Router node is connected to each gateway, and multiple gateway devices are connected together via an Ethernet backbone.



In this example, each gateway configures a Border-Router via 6LoWPANd to be on a separate /64 IPv6 prefix. Each gateway then uses Radvd to advertise a route to the relevant /64 prefix to the Ethernet network. Other devices in the Ethernet network set up routes to each prefix via the relevant gateway using the router advertisements sent by Radvd. The following are the openWrt configuration files necessary to set this configuration.

Gateway 1: /etc/config/6LoWPAN

```
config 6LoWPANd 'tun0'
    option ignore '0'
    option tty '/dev/ttyUSB0'
    option baudrate '1000000'
    option interface 'tun0'
    option activityled 'none'
    option channel '11'
    option pan '0x1234'
    option jennet '0x11111111'
    option prefix 'fd04:bd3:80e8:2::'
    option qos_enable '0'
    option secure '1'
    option jennet_key '::1'
    option auth_scheme '1'
    option radius_ip 'fd04:bd3:80e8:1::1'
```

Gateway 1: /etc/config/radvd

```
config 'interface'
        option 'interface' 'lan'
        option 'AdvSendAdvert' '1'
        option 'ignore' '0'
        option 'IgnoreIfMissing' '1'
        option 'AdvSourceLLAddress' '1'
        option 'AdvDefaultPreference' 'medium'
```

```
config 'prefix'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvOnLink' '1'
        option 'AdvAutonomous' '1'
        list 'prefix' 'fd04:bd3:80e8:1::/64'
        option 'AdvRouterAddr' '1'

config 'route'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvRouteLifetime' '3600'
        option 'AdvRoutePreference' 'medium'
        list 'prefix' 'fd04:bd3:80e8:2::/64'
```

Gateway 2: /etc/config/6LoWPAN

```
config 6LoWPANd 'tun0'
    option ignore '0'
    option tty '/dev/ttyUSB0'
    option baudrate '1000000'
    option interface 'tun0'
    option activityled 'none'
    option channel '11'
    option pan '0x1234'
    option jennet '0x11111111'
    option prefix 'fd04:bd3:80e8:3::'
    option qos_enable '0'
    option secure '1'
    option jennet_key '::1'
    option auth_scheme '1'
    option radius_ip 'fd04:bd3:80e8:1::1'
```

Gateway 2: /etc/config/radvd

```
config 'interface'
        option 'interface' 'lan'
        option 'AdvSendAdvert' '1'
        option 'ignore' '0'
        option 'IgnoreIfMissing' '1'
        option 'AdvSourceLLAddress' '1'
        option 'AdvDefaultPreference' 'medium'

config 'route'
        option 'ignore' '0'
        option 'interface' 'lan'
        option 'AdvRouteLifetime' '3600'
        option 'AdvRoutePreference' 'medium'
        list 'prefix' 'fd04:bd3:80e8:3::/64'
```

In this case, gateway 1 acts as a master. It advertises the IPv6 prefix to the Ethernet network and gateway 2 configures itself as a device within this prefix. Gateway 1 also acts as the RADIUS server (on address fd04:bd3:80e8:1::1). Every other gateway is configured to use this master gateway as its RADIUS server. In this way, a centralised white-list may be maintained for the whole system. Alternatively, each gateway may have a separate white-list by setting each gateway's configured RADIUS IPv6 address to its Ethernet IPv6 address.

This may be useful in controlling the commissioning process to force nodes to join a particular network. Otherwise, the nodes will tend to join the smallest available network.

6.3 Configuration of IPv6 Routing to /64 Prefixes on Linux

In order for IPv6 devices to be able to send packets to each other, they require a “route” to be set. The route may be set up manually by the network administrator, or automatically using the method built into IPv6 called “router advertisement”.

Routes must be set up on each client device and gateway in the system so that they know which gateway is responsible for forwarding packets into the destination JenNet-IP IEEE802.15.4 WPAN. To allow devices within a JenNet-IP IEEE802.15.4 WPAN to connect to devices in other JenNet-IP IEEE802.15.4 WPANs, the connected gateway must have a route configured to the destination prefix. Likewise, the target gateway must have a route back to the source prefix in order to route responses.

In general, in a multiple gateway configuration, every gateway and client device must have a route configured to each /64 prefix that is in use.

6.3.1 Manual Configuration

On Linux, a route to a /64 prefix may be configured manually using the command:

```
# ip -6 route add <prefix>/64 via <gateway link local address> dev <network device>
```

where:

- `<prefix>` is the prefix of the required IEEE802.15.4 WPAN
- `<gateway link local address>` is the link local IPv6 address of the gateway that has the required Border-Router node attached
- `<network device>` is the name of the network device through which this network is accessible - for example, “eth0”

This should be repeated for each /64 prefix in use.

6.3.2 Automatic Configuration

In order for Linux (including openWrt running on each gateway device) to accept advertised routes to /64 prefixes, some additional configuration is necessary. In the default configuration, Linux will accept only advertised default routes (routes to the /0 prefix). This is because it is assumed that automatic configuration is used only for the simplest of cases (the default route) and anything more complex must be configured manually. Changes are required to the kernel configuration to allow reception of non-default router advertisements, and a runtime configuration change is necessary to enable this.

The required kernel change is to ensure that the kernel configuration options CONFIG_IPV6_ROUTER_PREF and CONFIG_IPV6_ROUTE_INFO are enabled. In openWrt, these options are set via the default kernel configuration and are disabled by default. To enable them, edit the file:

backfire/target/<target>/backfire/target/linux/generic/config-3.3

Ensure that the above two options are enabled and rebuild the image.

Once the running kernel has these options enabled, a new **sysfs** file is created at:

/proc/sys/net/ipv6/conf/<interface>/accept_ra_rt_info_max_plen

By default, this file has the value 0, meaning that the maximum prefix length that the kernel will accept from router advertisements is 0, i.e. a default route. Setting this value to 64 allows the kernel to accept routes to the /64 prefixes advertised by radvd on each gateway.

When the kernel is set up to forward IPv6 packets between interfaces (as is always the case on an Ethernet to JenNet-IP gateway), an additional change is necessary to accept router advertisements containing a default route – it is necessary to change the value in the following **sysfs** file:

/proc/sys/net/ipv6/conf/<interface>/accept_ra

This is usually set to 1 to accept router advertisements. In order to accept router advertisements while forwarding is enabled, this must be set to 2.

Each of these changes to **sysfs** files may be done automatically at each boot by adding the options to the **/etc/sysctrl.conf** file.

6.4 Multicasts in a Multiple Gateway System

Each gateway has an instance of the multicast forwarding daemon mrd6 running (see Section 4.3). On each gateway, this daemon builds a list of multicast groups that exist in the connected JenNet-IP IEEE802.15.4 network. Using this knowledge, it selectively forwards multicast packets that arrive on the Ethernet interface into the JenNet-IP network. This does not work in reverse, however, and mrd6 does not send group membership information or multicast packets that originate within the JenNet-IP IEEE802.15.4 network out onto the Ethernet. Therefore, global and group control are limited to being originated outside of the JenNet-IP IEEE802.15.4 networks. This is an advantage in that multicasts are constrained within each network, limiting the overall traffic, but it means that a device within one of the networks cannot broadcast to devices in another network. Therefore, the network topology should be designed such that controller devices are located within the same network as the devices they control.

Global and group controls are always available to other client devices on the Ethernet network, as these will be forwarded across the gateway by mrd6.

7 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.

7.1 Compiling OpenWrt

7.1.1 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
- Subversion and git
- Ncurses/ncurses-devel

7.1.2 Setup 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 in which to work. Change to this directory and extract the openWRT source package using the following command:

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

Change directory to the target that you want to build (in this case, we are building for the Linksys WRT160NL) using the following command:

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

Set up 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, copy the NXP JenNet-IP Border-Router modifications and a default configuration into the checkout. The NXP JenNet-IP Border-Router modifications to the OpenWrt tree are held in 4 locations:

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

7.1.3 Install JenNet-IP Modifications to OpenWrt

Once the OpenWrt source tree has been checked out and modified with the NXP JenNet-IP Border-Router 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 the 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/
```

7.1.4 Build OpenWrt

Once the source tree has been prepared and the NXP JenNet-IP Border-Router 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 to change this 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 may be found in the directory:
openWRT/targets/<target>/backfire/bin/<architecture>

The root file-system for the target may 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 they could be mounted as an NFS root file-system for quick debugging.

Revision History

Version	Notes
1.0	First release
1.1	Formatting and minor changes made
1.2	Updated for JenNet-IP v1.2

Important Notice

NXP Semiconductors

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

www.nxp.com