MPTN Gateway Guide and Design Document

Updated on May 1, 2015

[Guide to enable a ZWave Gateway](#_661w13k080i0)

[Build Darjeeling VM](#_7qlawuoxhjkp)

[Run the WuKong Master](#_ksdta95uu550)

[Run the WuKong Zwave Gateway](#_25s9i4sbw8te)

[Deploy Application to Board](#_77g5frxdwl14)

[Multi-Protocol Transport Network Gateway Design](#_frvp0mtmy9dv)

[Goal](#_qjoadni8nh7p)

[Network Architecture](#_vbf5cjwvlxqg)

[Definitions](#_azmhv3tdy1no)

[Multi-Protocol Transport Network](#_q9yu7b9j7ois)

[Device](#_do2f4ab3n5c1)

[Node](#_4esbrbpfni1x)

[Gateway](#_hujeenuit1oz)

[Address](#_xrylg4leqenq)

[MPTN ID](#_5g35g7d9srxb)

[Routing Algorithm](#_1cl279jy887c)

[Packet Format](#_qsc5b6qw3myl)

[Software Architecture](#_1mifxouut9gw)

[Master](#_xc8911g85b2l)

[Node](#_2kgjesfpvzxu)

[Gateway](#_rcc8iuzat4zz)

[Message Types](#_k8iuepxe4rd6)

[ID](#_ejdkwkmo7ryr)

[RPC & FWD](#_8tuiew9ijph3)

[GWID](#_xdhn4udgcq6q)

[RT](#_jfkkbt1nzfss)

# Guide to enable a ZWave Gateway

*from Guide to enable WK on Intel Galileo*

## Build Darjeeling VM

* 1. Install the toolchain following the Wukong Release 0.2 Installation Guide (https://docs.google.com/document/d/1nXU3Mb1LFl9Sxozq\_JoVgQSq0gHwUK3WDLNKn21-tOc/edit) if you have not done so.
  2. Download the Source Code.  
     git clone https://github.com/wukong-m2m/wukong-darjeeling
  3. Pull and switch to release0.3 branch  
     git checkout release0.3
  4. Build the Infuser.  
     cd <dj root>/src/infuser  
     ant
  5. Modify the USB port setting of WuNode  
     cd <dj root>/src/  
     gedit settings.xml  
       
     Change the value of USB port to be like /dev/ttyUSB0 or /dev/cu.usbserial-A9CNZHXH
  6. Compile and Upload the Darjeeling to the WuNode  
     cd <dj root>/src/config/wunode  
     ant  
     ant avrdude

Next page

## Run the WuKong Master

Note that this part of guide follows the Wukong Release 0.2 installation guide, if anything changes in the future, please refer to that guide for any updates.

* 1. Modify the configuration file  
     cd <dj root>/wukong/config/  
     cp master.cfg.dist master.cfg  
     gedit master.cfg  
       
     Change the WKPFCOMM\_AGENT  
     WKPFCOMM\_AGENT = GATEWAY
  2. Run the master  
     cd <dj root>/wukong/master  
     python master\_server.py

Next page

## Run the WuKong Zwave Gateway

* 1. Open another terminal window either on the same PC (or another PC with WuKong toolchain installed as the link above and download the code from github)
  2. Install netifaces python module  
     sudo pip install netifaces (for linux)  
     sudo easy\_install netifaces (for MacOSX)
  3. Install pyzwave by compiling  
     cd <dj root>/wukong/tools/python/pyzwave  
     sudo python setup.py install
  4. Modify the configuration file  
     cd <dj root>/wukong/gateway  
     gedit gtwconfig.py  
       
     Make sure the  
     ENABLE\_AUTONET = False  
     ENABLE\_MONITOR = False  
     ENABLE\_PROGRESSION = False  
       
     Make sure the port consistent with the one used by Zwave controller:  
     TRANSPORT\_DEV\_ADDR = config.get('DEV\_ADDR', '/dev/ttyACM0')  
       
     Make sure the type is ‘zwave’:  
     TRANSPORT\_DEV\_TYPE = config.get('GATEWAY\_TYPE', 'zwave')
  5. Run the gateway  
     cd <dj root>/wukong/gateway  
     python start\_gateway.py

Next page

## Deploy Application to Board

* 1. Add WuNode to the master.  
     Open a browser and go to <http://localhost:5000>
  2. Click on Node Editor → +/- Zwave Nodes → Add nodes
  3. On WuNode to be added, press the button on the back
  4. On the master page, you will see the node being assigned with a new ID, click on stop if you finish adding the node(s).
  5. Build your own application.
  6. Click map and then deploy to upload the application to WuNodes.

End

# Multi-Protocol Transport Network Gateway Design

## Goal

To enable the communication among devices using heterogeneous network protocols, a transport layer gateway is designed in WuKong framework to bridge the connections of different networks. The gateway is called *Multiple-Protocol Transport Network gateway,* MPTN for short. MPTN gateway is a software component to be deployed on the devices having more than one network interface. (At time of this writing, the gateway assumes IP network is one of the network interface.)

Note that Master does not need to manage the transportation to and from devices which may affect the running thread of web server.

## Network Architecture

Gateway stands at the Application layer between the profile framework and the Transport layer where the standard protocols like TCP/IP, Zigbee, or Z-Wave resides.



## Definitions

### Multi-Protocol Transport Network

Currently, MPTN supports three protocols: TCP/IP (or IP), Zigbee (or ZB), and ZWave (or ZW).

### Device

The term device indicates all the computing machines running WuKong-related VM or server including Master, gateway, WuDevice, Arduino, Taroko, or Galileo.

### Node

For the devices other than Master and gateway, we call them the nodes.

### Gateway

A gateway is a device transmitting the cross-protocol packet like IP to and from ZW, or ZB to and from ZW. By definition, It should have at least two network or radio interfaces (one must be the member of TCP/IP suite, e.g. Ethernet or WiFi to connect between Master or other gateways and itself).

However, software implementation of gateway would be one process managing only two interfaces, e.g. IP + ZB or IP + ZW. To support more than two protocols, more processes would be needed. Note that the software also assumes the ZB or ZW interface is the controller or the coordinator, respectively.

All gateways are transparent and should always exist on the network. They do not make routing decision for IP, ZB, or ZW after the packet is sent to the underlying Transport layer.

### Address

Address (Addr) is the unique address of each radio or network interface of the supported protocols on a device.

The lengths of Addr of ZW, ZB, and IP are one, two(short ZB address), and four(IPv4) bytes, respectively. ZB and ZW controllers (or coordinator) assign ZB and ZW addresses, and external DHCP server assigns IPv4 addresses.

### MPTN ID

MPTN ID (ID) is the unique address of each device on the MPTN.

It has four bytes and the same notation as IPv4 address (1.2.3.4/8) which composed of the network part and host part except that MPTN uses the IP network segment 0.0.0.0/8 to include ZW and ZB devices. The address is embedded into the lower bytes of ID or exactly the same for IP network address. The network prefix of a ID is decided by Master of which the ID would always be 0.0.0.0 (0x0).

Since ZW devices have one-byte addresses, they would reside in the block 0.0.W.0/24 where the byte W in the 24-bit network prefixes (0.0.W) is assigned by Master. Note that ZW controller would always be ID 0.0.W.1 with different W, and the same Addr 0x1.

Since ZB devices have two-byte short addresses after joining the ZB network, they would reside in the block 0.B.0.0/16 where the byte B in the 16-bit network prefixes (0.B) is assigned by Master. Note that ZB coordinator would always be ID 0.B.0.0 with different B, and the same Addr 0x00.

### Routing Algorithm

So far the MPTN supports only one dummy gateway routing algorithm. All devices would compare the prefix of destination ID to its own prefix in order to determine the next hop and extract the Addr to send (either the address of another device on the same network block or the address of gateway if the destination ID belongs to different networks)

### Packet Format

The format of the packet transmitted between the gateway and the other devices with VM installed consists of a four-byte destination ID, a four-byte source ID, a one-byte message type, and the payload from the upper Profile Framework. Because the payload length limitation of the minimum of the underlying network protocols, the payload of MPTN is 40 bytes long.

The format of the packet transmitted between the gateway and other gateways or Master is similar to the above statement but for each packet transmission, a random session nonce and the length of the packet would be sent out first.

## Software Architecture

There are three roles on the MPTN: Master, gateway and node.

### Master

The python program “transportv3.py” implements the MPTN function on the Master, which contains a RPC agent to communicate with the gateways via RPC protocol to send a packet, add, delete, stop, poll, discover, get device type. There is another socket server in that agent listens to the TCP port 9010 to receive the packets from gateways.

### Node

The C program “routing\_gateway.c” implements the MPTN function in the VM, which compares the prefix and sends packets and asks for the ID from Master via the gateway.

### Gateway

The gateway is a message-oriented two-greenlets python program. From the guide above, the start\_gateway.py is the entry point of the gateway. It would first initialize the radio transport interface server and the main gateway TCP server. After that it would run both the servers as greenlets or green threads to process the messages come from either the radio transport interface or the TCP server after the gateway starts.

For example, the class ZWTransport in the “zwave.py” implements the function of init, send, receive, get device type, routing, discover, add, delete, stop, and poll for the ZWave controller.

The class Gateway in the “gtwclass.py” would use those functions to respond to Master’s RPC requests via the TCP socket.

There are two services in the gateway to handle two different message types: RPC service and ID service, each of which are the program “rpcservice.py” and “idservice.py” for message type RPC and ID, respectively.

The program “mptnUtils.py” defines the constants and tools used for MPTN and handles the packets between Master or other gateways.

## Message Types

Currently, different message categories of packets run on the MPTN: ID(ID), Gateway ID(GWID), Remote Procedure Call(RPC), Routing Table (RT) and Forward(FWD). There would be different situations to apply each of them: the initialization of a node, the initialization of a gateway, the RPC request of Master, updating routing table between master and gateway, and routing between devices.

### ID



The detailed packet format of IDREQ, IDACK is as follows:

| Destination ID (4 bytes) | Source ID (4 bytes) | Message Type (1 byte) | Payload

N2G 0xFFFFFFFF 0xNODEADDR GWDISCOVER

G2N 0xFFFFFFFF 0xGATEWYID GWOFFER [16 bytes UUID]

N2G 0x00000000 0xFFFFFFFF IDREQ [16 bytes UUID]

G2M 0xGATEWYID 0xNODE\_\_ID IDREQ [16 bytes UUID]

M2N 0xNODE\_\_ID 0x00000000 IDACK

M2G 0xFFFFFFFF 0x00000000 IDNAK

After the node learns a network address, it broadcasts GWDISCOVER to find the gateway.

0xFFFFFFFF indicates the lack of the ID on the node. The gateway would reply a GWOFFER with a 16-byte UUID which could be used to handle dynamic address issue.

Because some nodes may have limited computing power, both node and gateway could generate the UUID and the node could choose either the self-generated one or the received one to be the payload of IDREQ. Gateway generates the UUID by python function uuid.uuid4() which follows the UUID Version 4 specification. Node is suggested to use the same standard but not required. It would be fine for the node to use a external chip or random algorithm to get one.

IDREQ is sent by the new node to gateway with destination ID being Master and source ID being 0xFFFFFFFF. The gateway would check if the sender’s network address has been allocated. If True and UUID is the same as the one for that network address, gateway would reply IDACK directly without further sending IDREQ to the Master. If UUID is different, it means a different node and the node would get no response. If the address is not recognized, gateway would then replace the destination ID with the gateway ID and the source ID with a new node ID because gateway knows the new node’s network address and gets the new node’s ID by adding its prefix got from Master.

Master would check if UUID has ever been saved before. If not and the GATEWYID is valid and the NODE\_\_ID is also unseen, then Master add NODE\_\_ID into the records. If UUID is found in the record, Master would compare the ID from record with the NODE\_\_ID which should be the same one. IDACK will be returned to the original gateway if either the new NODE\_\_ID is inserted or it matches the ID in the record and then IDACK will be sent to the node by gateway. Gateway would get IDNAK if any of the test fails. However, IDNAK will be dropped by gateway instead of being sent to the node as IDACK is.

### RPC & FWD

The RPCCMD by Master and the RPCREP by gateway are defined as follows.

M2G 0xGATEWYID 0x00000000 RPCCMD [RPC JSON command]

G2M 0x00000000 0xGATEWYID RPCREP [RPC JSON response]

Supported functions are (send,getDeviceType,routing,discover,add,delete,stop,poll) which are implemented by underlying network protocol drivers such as zwave.py, udp.py, etc.

D2D 0xDESTNDID 0xSOURCDID FWDREQ [WKPF payload]

D2D 0xSOURCEID 0xDESTINID FWDACK []

D2D 0xSOURCEID 0xDESTINID FWDNAK [error\_message]

FWDREQ messages are used between every device, especially for WKPF messages. Only the gateway or Master would reply FWDACK or FWDNAK back to the sender which is either other gateways or Master but not nodes.

### GWID

G2M 0x00000000 0xOLDGWYID GWIDREQ [JSON]

M2G 0xGATEWYID 0x00000000 GWIDACK []

M2G 0xFFFFFFFF 0x00000000 GWIDNAK []

GWID is used for gateway initialization. If gateway first starts, the source 0xOLDGWYID would be 0xFFFFFFFF. JSON format is {'IFADDR':int, 'IFADDRLEN':int, 'IFNETMASK':int, 'PORT':int, 'UUID':a list with 16 unsigned chars}. 0xGATEWYID would be the confirmed or new allocated ID for gateway.

### RT

G2M 0x00000000 0xGATEWYID RTPING [SHA-512 hash of the routing table]

M2G 0xGATEWYID 0x00000000 RTPING [SHA-512 hash of the routing table]

G2M 0x00000000 0xGATEWYID RTREQ []

M2G 0xGATEWYID 0x00000000 RTREP [routing table JSON]

RTPING with a sha-512 hash of the routing table as the payload is sent out regularly by gateway to Master to query the latest routing table. Master would send back its hash using RTPING message. If the gateway finds the hash is different, RTREQ is sent out to get the table and the gateway gets RTREP with a json format of routing table as the following structure.

{“MY.AS.IP.FORMAT/NETMASK”:[“IP”:PORT], ...} (Note the netmask is an integer like /24)

Old pictures

