ZMDI
The Analog Mixed Signal Company

Application Note

Rev. 1.00 / August 2014

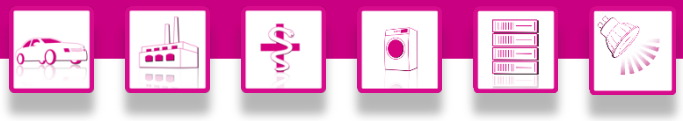
NetMA Protocol

ZWIR45xx Network Monitoring and Administration

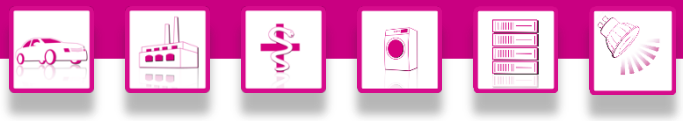
Automotive ASICs and Industrial ASSPs

Interface ICs

Multi-Functional and Robust



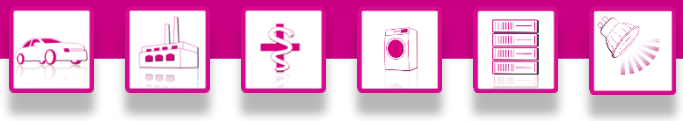
Contents



List of Figures

List of Tables

For more information, contact ZMDI via wpan@zmdi.com.



1 Requirements Notation

This document uses several words to indicate the requirements of ZMDI's NetMA implementation. The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*, *RECOMMENDED*, *MAY,* and *OPTIONAL* set in italic small caps letters denote requirements as described below:

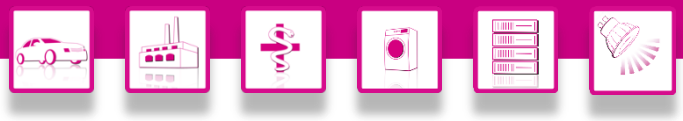
The words *MUST, SHALL, and REQUIRED* denote an absolute requirement of the implementation. Disregarding these requirements will cause erroneous function of the system.

The words *MUST NOT and SHALL NOT* mean that something is absolutely prohibited by the implementation. Disregarding these requirements will cause erroneous function of the system.

The words *SHOULD and RECOMMENDED* describe best practice, but there might be reasons to disregard it. Before ignoring the statement, the implications of ignoring it must be fully understood.

The words *SHOULD NOT and NOT RECOMMENDED* describe items that can impair proper behavior of the system if implemented. However, there might be reasons to choose to implement the item anyway. Implications of doing so must be fully understood.

The words *MAY and OPTIONAL* describe items that are optional. No misbehavior is to be expected when these items are ignored.



2 Introduction

Developing, maintaining, and using wireless mesh networks might, depending on the application and its working environment, be a difficult task. ZMDI's ZWIR45xx devices have been designed to be easily usable and to minimize administration and configuration effort. However, in some application scenarios and working environments, the ZWIR45xx might require additional attention to the configuration to achieve the optimal communication performance. Often such optimizations must be performed after the installation of the network in its working environment, as this environment might add different constraints due to its own characteristics. At this point, the application is typically installed on the module and changing it is not desirable. For this reason, ZMDI provides a protocol that can be used to analyze the network topology and to read and write network parameters. This protocol is called the NetMA protocol, which is an acronym for Network Monitoring and Administration protocol.

This document refers to the NetMA version 2 protocol (NetMA2), which is the first version providing official documentation. NetMA version 1 is still available as a separate library, but it is not recommended to be used for new designs. References to NetMA in this document always refer to NetMA version 2 if not denoted otherwise.

The NetMA protocol is provided in two separate firmware libraries, which *MAY* be included into the customer application to provide the desired NetMA functionality. Using this approach, NetMA functionality can be provided for every kind of application, independent from the application's functionality.

A brief overview of the functionalities provided by NetMA is given in sections 2.1 through 2.4. Protocol details are explained in section 3. Section 4 covers the topic of using NetMA directly from a ZWIR45xx device. In section 5, there is a discussion of security aspects that must be considered in conjunction with the NetMA protocol.

2.1. Parameter and Status Readout and Configuration

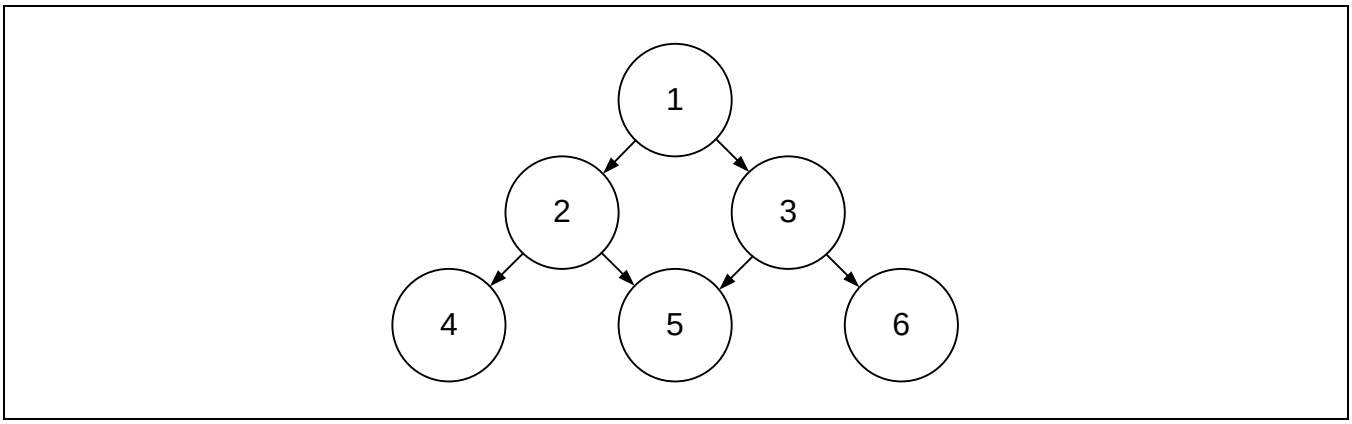
NetMA provides the functionality to read and write all parameters that can be changed via programming. In addition, invariant parameters and status values, such as firmware versions or transceiver statistics, can be obtained remotely using NetMA. For permanent parameter changes, NetMA provides functions to store the current parameter configuration to the ZWIR45xx's flash memory or to restore its default configuration.

For querying and setting parameters, NetMA provides a data format that allows fine-grain specification of the parameters that should be affected by a certain command. Fine-grain parameter specification is provided for Remote Parameter Request, Remote Parameter Response, and Remote Parameter Set packets. For these packets, a common parameter specification format is provided as described in section 3.4.

The Remote Parameter Store and Remote Parameter Defaults packets are used to store the current parameter configuration or to restore the default configuration. Note that both commands work on the whole set of parameters. Thus, when parameters are stored, all configured parameters are written into the ZWIR45xx's flash regardless of the source of parameter change. Stored parameters are automatically restored after a device reset. If a device has been reconfigured without storing the parameters, the last stored parameter configuration will be restored after a system reset or after waking up from Standby Mode.

Note: Take care when parameters affecting the physical or mesh networking behavior are changed. If such change should be performed in the whole network, the changes must be performed from the leaves to the trunk. Otherwise, single network nodes or whole branches of the network might be unreachable. For example, consider the effect of changing the radio frequency of the whole network shown in Figure 2.1. For correct operation, the change must be executed in nodes four to six first, followed by nodes two and three and finally node one. Changing node one first would result in the remaining nodes becoming unreachable because they would still be using the old frequency. Changing nodes two or three first would result in nodes four or six becoming unreachable, respectively.

Figure 2.1 Example Network Topology



The NetMA functionality dedicated to setting and storing parameters is provided as a separate library. This is to prevent users from opening potential security vulnerabilities, allowing attackers to change the network behavior or to break down network communication completely.

Important: When the NetMA parameter set and store library is included in a project, enabling IPSec is highly recommended for protection of the user's network. Refer to section 5 for further information regarding security.

2.2. Network Topology Detection

NetMA's Neighbor Request packets can be used to discover the network topology of the wireless mesh network. Neighbor Requests trigger the transmission of Neighbor Responses, which include lists of neighboring nodes to which direct communication without intermediate hops is possible. Together with the list of neighbors, Neighbor Responses include bi-directional link quality information. For the purpose of direct neighbor discovery, a node receiving a Topology Request packet sends out a special Remote Parameter Request that is handled by all neighboring nodes. The responses are collected by the node that received the Neighbor Request and after a specific timeout, a list containing the PAN addresses of all responding nodes is returned to the device that initiated the Neighbor Request.

The Neighbor Request packet allows specifying filter criteria, which can be used to limit the scope of the request; for example, to receive responses only from a specific device type. This feature can also be used to improve the reliability of the discovery process. This is accomplished by querying a node twice for its neighbors, using the same Query ID filter for successive requests.

2.3. Mesh Network Route Tracing

Sometimes it is necessary to know the exact route a packet travels through the network from one node to another. For this purpose, NetMA provides a route tracing functionality. Route tracing can be performed directly or indirectly. Direct tracing means that the node requesting the trace should be the starting point of the route as well. Indirect tracing means that the starting point of the route is different from the node requesting the trace. Consequently, traces performed from a typical computer must always be indirect.

Route tracing works on PAN addresses only. Thus, knowing the target's PAN address is required in order to trace a specific node. A node receiving a Trace Request packet appends its own PAN address along with link quality information for the received packet. Then it checks whether the requested target node equals its own PAN address. If it does, a Trace Response packet is sent to the initiator of the request, containing all hop information that has been appended during the tracing process. If not, the node checks whether there is route information toward the target PAN address available in its routing table. If so the node forwards the packet to the next hop, which is obtained from the routing table. Otherwise, the node sends a Route Fail packet, containing all hop information that has been appended during the tracing process toward this route.

2.4. Routing Table Analysis

The Routing Table Analysis debugging feature allows analyzing the contents of the routing table of each individual node. Using the routing table information, it can be determined offline which route would be used through the network for certain packets.

As the routing tables might be relatively large and the maximum number of bytes that can be transmitted with one IPv6 packet is limited in ZWIR45xx devices, large routing tables are fragmented for transmission. In order to obtain the complete routing table of a node, the first fragment should be requested first and the corresponding status flag in the Routing Table Response should be evaluated to determine if there are further fragments available.

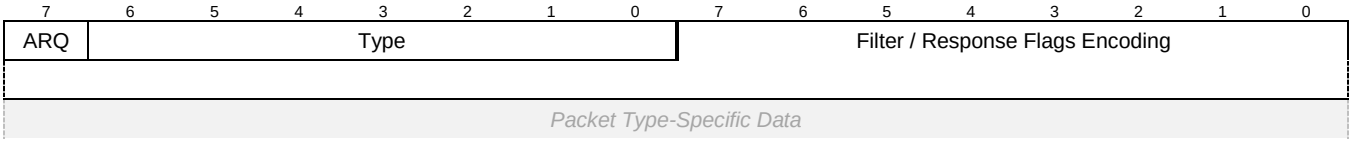
3 NetMA Protocol

3.1. NetMA Common Header

NetMA packets share a common header, which has a minimum length of two bytes. These two bytes define the packet type and a set of flags that control how the packet is processed by the receiver. Depending on the flag configuration, additional fields must be included in the header. For some packet types, some of the flags are not applicable and will be ignored. Refer to the documentation for the different packet types for details.

Note: Fields that are optional are indicated by an asterisk (*) in subsequent figures.

Figure 3.1 NetMA Common Header



ARQ

If this bit is set to 1, this flag signals that an acknowledgement is requested from the receiver of this packet.

Type

This field determines the packet type. A list of valid values for the ***Type*** field is given in section 3.1.1.

Filter / Response Flags

In NetMA requests, this variable-length encoding specifies the filter criteria, which can be used to limit the scope of a request in specific ways. The format of the filter field is specified in section 3.1.2.

In NetMA responses, this encoding contains flags, indicating the node's status. The format of the ***Response Flags*** field is specified in section 3.1.3.

3.1.1. Packet Types

The 7-bit wide ***Type*** field specifies the type of the packet and therefore determines what data is following the common NetMA header.

Table 3.1 Packet Type Overview

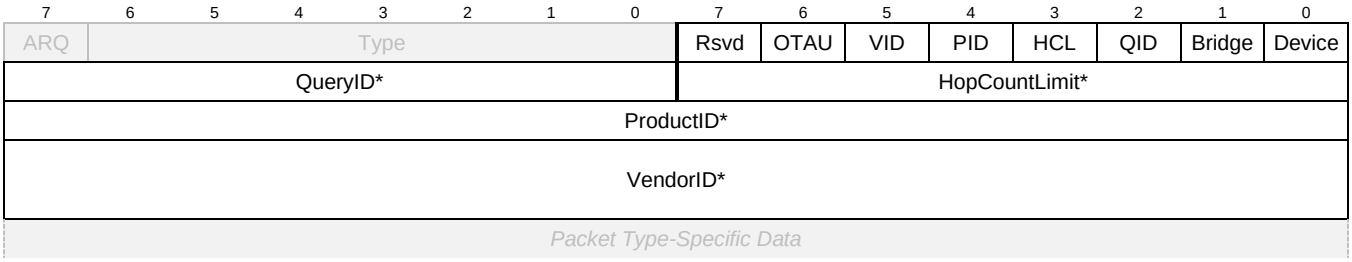
Type	Description	Refer to Section
00_{HEX}	Acknowledge	3.2
01_{HEX}	Reject	3.3
08_{HEX}	Remote Parameter Request	3.4.2
09_{HEX}	Remote Parameter Response	3.4.3
$0a_{HEX}$	Remote Parameter Set	3.4.4
$0b_{HEX}$	Remote Parameter Store	3.4.5
$0c_{HEX}$	Remote Parameter Defaults	3.4.6

Type	Description	Refer to Section
28_{HEX}	Reset Device	3.5
29_{HEX}	Reset TRX Statistics	3.6
10_{HEX}	Routing Table Request	3.7
11_{HEX}	Routing Table Response	3.8
18_{HEX}	Neighbor Request	3.9
19_{HEX}	Neighbor Response	3.10
20_{HEX}	Trace Request	3.11
21_{HEX}	Trace Response	3.12
22_{HEX}	Trace Fail	3.12

3.1.2. Filters

The NetMA Protocol provides request filters for different purposes. The primary purposes are limiting the number of responses received for a request sent to a multicast address and limiting the scope of the requests to nodes meeting specific constraints.

Figure 3.2 Common Header – Filter Fields



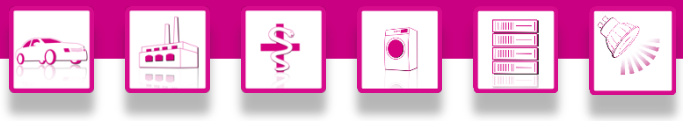
Rsvd

Fields marked with Rsvd are reserved for future use. ***Rsvd*** fields *MUST* always be set to 0 by the sender and *MUST* be ignored by the receiver of a packet.

OTAU

If this flag is set to 1 in a request, it signals that the receiver should process this packet only if it is enabled for the Over-The-Air-Update (OTAU) feature and ignore it otherwise. If the flag is cleared, each device *SHALL* process the packet regardless of its OTAU configuration.

In responses, this flag shows the OTAU configuration of the responding device. If OTAU is enabled, the flag is set to 1; otherwise it is cleared.



VID

If set to 1 in a request, this flag signals that the receiver *SHOULD* process this packet only if its Vendor ID is equal to the ***VendorID*** field included in this packet. The ***VendorID*** field *MUST* be included in the packet when this flag is set and *MUST* be omitted when the flag is cleared.

Important: Responses *MUST* always have this flag cleared.

PID

If set to 1, this flag signals that the receiver *SHOULD* process this packet only if its Product ID is equal to the ***ProductID*** field included in this packet. The ***ProductID*** field *MUST* be included when this flag is set and *MUST* be omitted when the flag is cleared.

Important: Responses *MUST* always have this flag cleared.

HCL

If set to 1, this flag signals that the receiver should process this packet only if the number of hops the packet has taken to reach the receiver is lower than or equal to the value specified in the ***HopCountLimit*** field. The ***QueryID*** and ***HopCountLimit*** fields *MUST* be included in the packet when this flag is set, and both fields *MUST* be omitted when both HCL and QID are cleared.

Important: Responses *MUST* always have this flag cleared.

QID

If set to 1, this flag signals that the receiver should process this packet only if it has not already processed a query with the same ***QueryID*** field from the same sender. The ***QueryID*** and the ***HopCountLimit*** fields *MUST* be included in the packet when this flag is set, and both fields *MUST* be omitted when HCL and QID are both cleared.

Important: Responses *MUST* always have this flag cleared.

Bridge

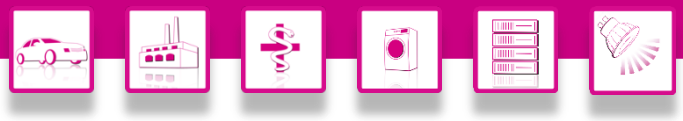
If this flag is set to 1 in a request, it signals that the receiver *MUST* process the packet when it is configured in Gateway Mode. If the device is a gateway and the ***Bridge*** flag is cleared, the packet *SHOULD* be silently ignored.

Responses have this flag set according to the device configuration of the responding device. If the responding device is configured in Gateway Mode, the ***Bridge*** flag is set to 1. Otherwise this flag is cleared.

Device

If this flag is set to 1 in a request, it signals that the receiver *MUST* process the packet when it is configured in Device Mode. If the receiver is a device and the ***Device*** flag is cleared, the packet *SHOULD* be silently ignored.

Responses have this flag set according to the device configuration of the responding device. If the responding device is configured in Device Mode, the ***Device*** flag is set to 1. Otherwise this flag is cleared.



QueryID

This field contains a numeric Query ID that is used to identify repeated NetMA requests from the same source. The ***QueryID*** field *MUST* be included in the packet together with the ***HopCountLimit*** field if the QID flag and/or HCL flag is set to 1.

HopCountLimit

This field contains the maximum number of hops that a packet is allowed to travel through the network to its receiver. Receivers with longer distances *MUST* silently ignore the request. The ***HopCountLimit*** field *MUST* be included in the packet together with the ***QueryID*** field if the QID flag and/or HCL flag is set to 1. The ***HopCountLimit*** *MUST NOT* be used in NetMA responses.

ProductID

This field specifies the Product ID to match against the receiver's Product ID. Receivers *MUST* ignore a request when their Product ID does not match the ***ProductID*** field exactly. The ***ProductID*** field *MUST* be included in a packet when the PID flag is set to 1. If the PID flag is cleared, the ***ProductID*** field *MUST* be omitted.

VendorID

This field specifies the Vendor ID to match against the receiver's Vendor ID. Receivers *MUST* ignore a request when their Vendor ID does not match the ***VendorID*** field exactly. The ***VendorID*** field *MUST* be included in a packet when the VID flag is set to 1. If the VID flag is cleared, the ***VendorID*** field *MUST* be omitted.

3.1.2.1. Hop Count Limit

This filter limits the number of responses on a request to those nodes that are within the hop-count limit distance. If the hop count limit is set to 0, only direct neighbors of the node sending the request will respond. This helps reducing network congestion and therefore increases the probability of successful delivery of the response. When multicast NetMA queries are sent into large networks, the Hop Count Limit filter *SHOULD* be used in conjunction with the Query ID in the following way: Send the first request with the specific Query ID and the ***HopCountLimit*** field set to 1. Continue querying the network with the ***HopCountLimit*** field increased by one, using the same Query ID. This procedure *SHOULD* be repeated until no responses from remote devices are received.

3.1.2.2. Query ID

The Query ID allows sending multiple equal requests into the network without triggering responses from nodes that have already successfully responded on the request. It is recommended to use this feature in all types of broadcast requests. Broadcast requests should be sent at least twice, carrying the same Query ID. The second request will trigger a response only from nodes that have not successfully responded on the previous request.

ZWIR4512 devices are only capable of storing one Query ID along with the sender's IPv6 address. Thus, if between two packets with the same Query IDs from the same sender, another packet with the same Query ID is received from a different sender, the second request from the original sender will be processed regardless of the status of the first response.

3.1.2.3. OTAU

This flag limits the scope of a request to nodes that have the Over-the-Air-Update (OTAU) feature enabled. Nodes without OTAU support will not respond on requests with the OTAU flag set.

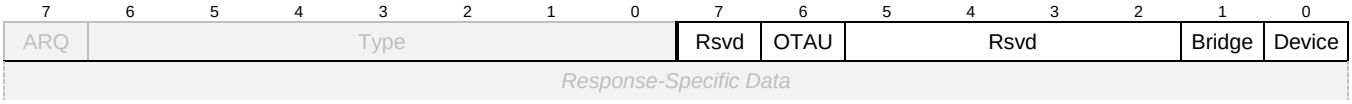
3.1.2.4. Product and Vendor IDs

This filter limits the scope of a request to nodes with a specific Vendor ID and/or Product ID. Nodes that receive a request with the ***VID*** and/or ***PID*** fields set will only process the request if their Vendor ID and/or Product ID match the values given in the ***VendorID*** and/or ***ProductID*** filter fields.

3.1.3. Response Flags

NetMA responses carry status information about the responding node in the ***Response Flags*** field. The ***Response Flags*** field is always 1 byte long. The format of the field is shown in Figure 3.3.

Figure 3.3 Common Header – Response Flags



Rsvd

Reserved for future use – always set to 0.

OTAU

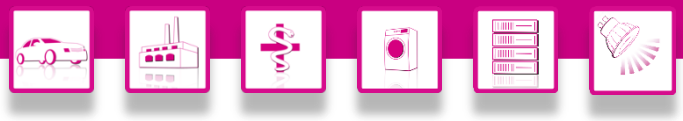
Status flag, indicating whether the node is OTAU enabled or not.

Bridge

Status flag, indicating that the responding node is configured in Gateway Mode.

Device

Status flag, indicating that the responding node is configured in Device Mode.



3.2. Acknowledge Packet

Acknowledge packets are used to confirm the reception of a request or response. Acknowledges are sent only if explicitly requested by the corresponding request or response. Typically acknowledges are only requested for response packets and request packets that do not send a response since for the other request packets the actual response can be considered as the acknowledge. An acknowledge is requested by asserting the ARQ bit in the NetMA Common Header. When a packet is received with the ARQ flag set, the receiver *MUST* immediately send the corresponding acknowledge. Acknowledges do not contain acknowledge specific data.

Response packets sent from a NetMA node typically require acknowledgement. If no acknowledge is received within one second, the Response packet will be repeated up to two times, seeking confirmation of successful delivery. If no acknowledge is received after three attempts, the packet delivery is considered to be failed and the node sending the response can respond on a second request carrying the same Query ID. Upon successful response delivery (acknowledge has been received), the device will not respond on a request carrying the same Query ID as the previous one.

Figure 3.4 Acknowledge Packet Layout

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ARQ	Type							Rsvd	Request Type						

ARQ

This bit *MUST* be set to 0 for acknowledge and reject packets.

Type

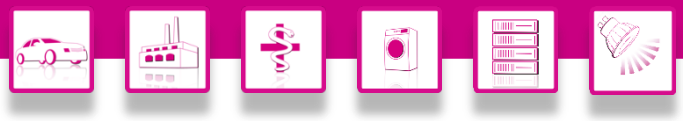
For acknowledge packets, this field *MUST* be set to 0.

Rsvd

Reserved for future use – always 0.

Request Type

This field contains the value of the ***Type*** field triggering the request.



3.3. Reject Packet

Reject packets are used to inform the sender of a packet about reception of a packet with invalid contents or a packet that could not be processed for other reasons. Reject packets are sent regardless of the value of the ARQ flag.

Figure 3.5 Reject Packet Layout

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
ARQ	Type							Rsvd	Reject Reason						

ARQ

This bit *MUST* be set to 0 for Acknowledge and Reject packets.

Type

For Reject packets, this field *MUST* be set to 1.

Rsvd

Reserved for future use – always 0.

Reject Reason

This field indicates the reject reason. The specific ***Reject Reason*** values are documented in the corresponding requests.

3.4. Remote Parameter Packets

NetMA provides a set of packet types used to obtain and change node configurations remotely. The packets provided for this purpose are Remote Parameter Request, Remote Parameter Response, Remote Parameter Set, Remote Parameter Store, and Remote Parameter Default. Most of these packets include a Parameter Specification controlling to which parameters the request or response is dedicated. The Parameter Specification format is defined in section 3.4.1 followed by the documentation for the actual packet types using this format.

3.4.1. Remote Parameter Specification

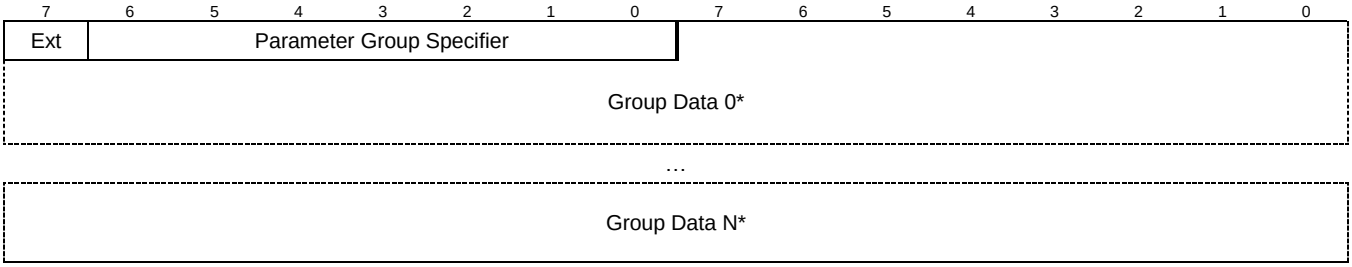
The Remote Parameter Specification is a variable length encoding that is included in Remote Parameter Request, Remote Parameter Response, and Remote Parameter Set packets. It allows fine grain specification of the parameters being processed during a transaction. Due to the fine granularity, it is possible to maintain compatibility through different network stack and NetMA protocol versions, even if the number of parameters is changed. Each parameter is assigned to a Parameter Group.

The remote parameter specification is split into three generic components:

1. Parameter Group Specifier with extension flag
2. Parameter Specifier with extension flag
3. Parameter Value List

The first component is a bit field controlling which Parameter Groups are included in the Remote Parameter Specification (see Figure 3.6). A ***Remote Parameter Specification*** *MUST* include at least one ***Parameter Group Specifier***. The second component is the ***Parameter Specifier*** bit field, which is used to select single parameters from a group. There *MUST* be at least one ***Parameter Specifier*** field for each Parameter Group that is enabled in the ***Parameter Group Specifier***. The last component is the ***Parameter Value List***. This component is only used in Remote Parameter Responses and Remote Parameter Set packets and contains parameter values. The Remote Parameter Request uses only the first two components to control which parameter values are requested.

Figure 3.6 Remote Parameter Specification Format



Ext

In the Remote Parameter Specification, this field is reserved for future use and *MUST* be set to 0.

Parameter Group Specifier

This bit field indicates which parameter groups are included in the ***Remote Parameter Specification*** encoding. The presence of a parameter group is indicated by a 1 in the bit mask. Refer to Table 3.2 for an overview of available Parameter Groups and the values within the groups.

Group Data

This is a variable length encoding, identifying which parameters of a particular group are addressed and the corresponding values if applicable. Figure 3.7 gives the format for the ***Group Data*** field.

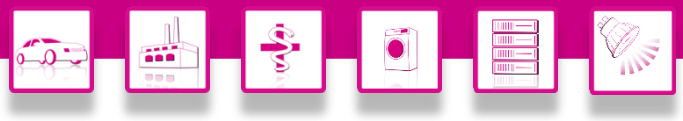
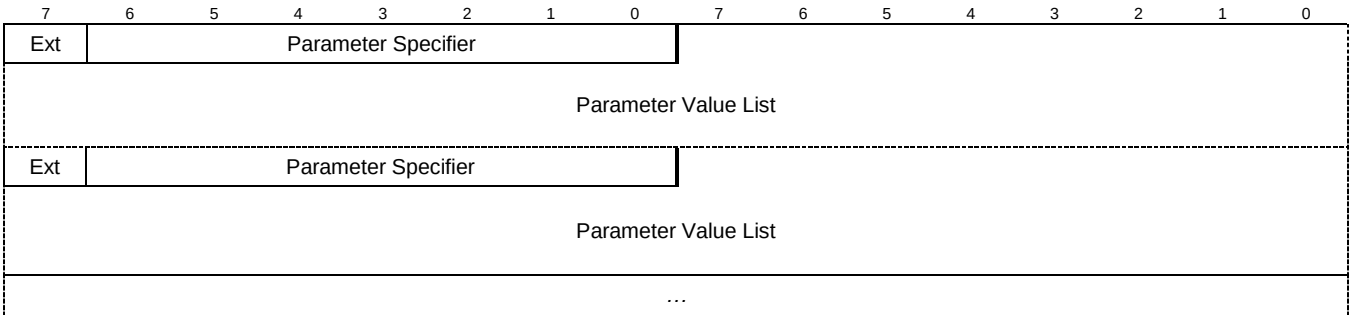


Figure 3.7 Group Data Encoding used in Remote Parameter Specification



Ext

In the Group Data encoding, this field is used when a parameter group contains more than 7 different parameters. Setting this field to 1 indicates the presence of a further ***Parameter Specifier*** field after the ***Parameter Value List*** belonging to the ***Parameter Specifier*** succeeding the ***ext*** field. The extended Parameter Specifier fields have their own ***Ext*** fields to indicate the presence of any further parameters.

Parameter Specifier

This bit field specifies which parameters are addressed by the packet. Each single parameter has its own indicator bit as defined in the ***Flag*** column in Table 3.2.

Parameter Value List

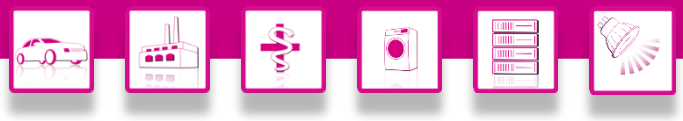
The ***Parameter Value List*** encoding contains the data values for the parameters specified by the ***Parameter Specifier***. The values are chained one after each other without padding. The parameters are ordered according to their corresponding position in the ***Parameter Specifier*** field from LSB to MSB.

The presence of the ***Parameter Value List*** encoding depends on the Remote Parameter Packet type. The ***Parameter Value List*** *MUST* be included in Remote Parameter Set and Remote Parameter Response packets. It *MUST NOT* be included in Remote Parameter Request packets.

Table 3.2 Parameter Groups and Parameters

See important notes at the end of this table.

Group	Flag[1]	Description	R/W	Bytes	Default Value
01	**Generic Parameter Group**				
	01_{HEX}	PAN Identifier	r/w	2	CAAC
	02_{HEX}	PAN Address	r/w	8	n/a
	04_{HEX}	Vendor ID	r	4	n/a
	08_{HEX}	Product ID	r	2	n/a
	10_{HEX}	Firmware Version	r	4	n/a
	20_{HEX}	Library Version	r	4	n/a

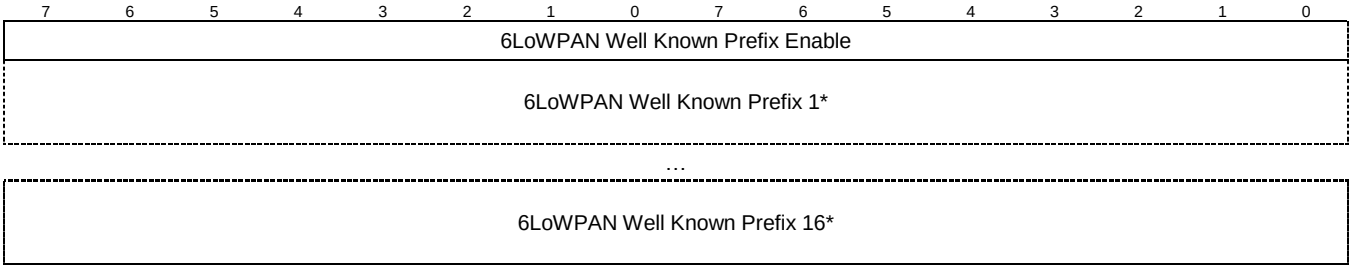


Group	Flag[1]	Description	R/W	Bytes	Default Value
02	**TRX Statistics Group**				
	01_{HEX}	Transmitted Byte Count	r	4	n/a
	02_{HEX}	Transmitted Packet Count	r	4	n/a
	04_{HEX}	Received Byte Count	r	4	n/a
	08_{HEX}	Received Packet Count	r	4	n/a
	10_{HEX}	Transmission Failure Count	r	4	n/a
	20_{HEX}	Duty Cycle	r	4	n/a
04	**Mesh Network Parameters Group**				
	01_{HEX}	Route Timeout	r/w	2	3600 s
	02_{HEX}	Routing Table Size	r/w	2	8
	04_{HEX}	Maximum Hop Count	r/w	1	4
	08_{HEX}	Route Maximum Fail Count	r/w	1	3
	10_{HEX}	Route Request Minimum Link RSSI	r/w	1	-128 dBm
	20_{HEX}	Route Request Minimum Link RSSI Reduction	r/w	1	0 dBm
	40_{HEX}	Route Request Attempts	r/w	1	3
08	**Physical Layer Parameters Group**				
	01_{HEX}	Transmit Power	r/w	1	0 dBm
	02_{HEX}	Channel	r/w	1	0 (868.3MHz)
	04_{HEX}	Modulation	r/w	1	0 (BPSK)
10	**Network Layer Parameters Group**				
	01_{HEX}	Neighbor Reachable Time	r/w	2	3600 s
	02_{HEX}	Neighbor Cache Size	r/w	1	8
	04_{HEX}	Maximum Socket Count	r/w	1	8
	08_{HEX}	Do Duplicate Address Detection	r/w	1	1
	10_{HEX}	Do Router Solicitation	r/w	1	1
	20_{HEX}	Do Address Auto-configuration	r/w	1	1
	40_{HEX}	Neighbor Retransmit Time	r/w	2	3000 ms
	1 - 01_{HEX}	6LoWPAN Well-Known Prefix Configuration encoding	r/w	Var[2]	Link-Local Prefix
	1 - 02_{HEX}	IPv6 Address Configuration encoding	r/w	Var[3]	Autoconfigured

1) Flags with three digits refer to extended parameter specifications.
2) Refer to the 6LoWPAN Well Known Prefix Encoding specification below.
3) Refer to the IPv6 Address Encoding specification below.

For efficiency reasons, the device's 6LoWAPN well-known prefix configuration (Group 10, Flag 1-01) and its IPv6 address configuration (Group 10, Flag 1-02) are encoded in the formats specified below. The encoding allows specifying the particular address or prefix items that should be changed with a Remote Parameter Set request and specifies which items are populated in Remote Parameter Responses. Using this encoding, NetMA provides support for up to 16 different IPv6 addresses and 16 different 6LoWPAN well-known prefixes. The format of the ***6LoWPAN Well Known Prefix*** encoding and the ***IPv6 Address*** encoding is shown in Figure 3.8 and Figure 3.9, respectively.

Figure 3.8 6LoWPAN Well Known Prefix Encoding in Parameter Specifications



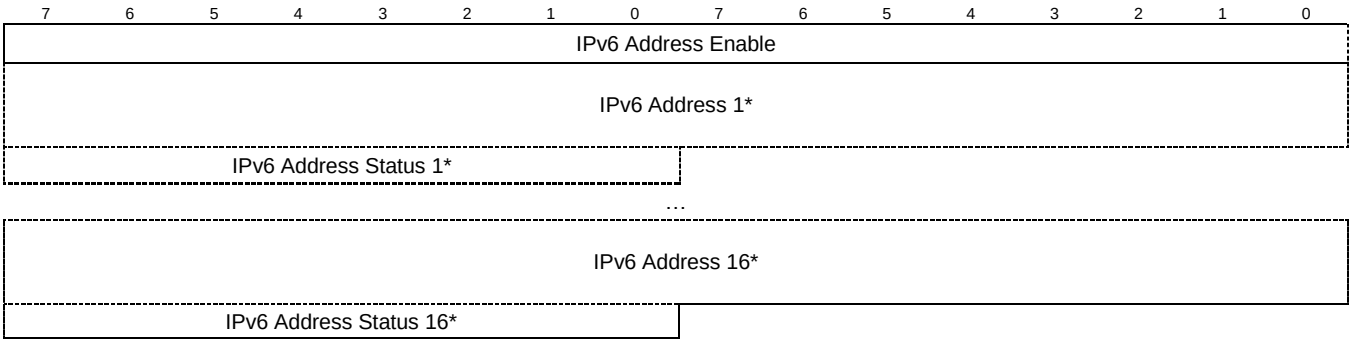
6LoWPAN Well Known Prefix Enable

This bit field determines which 6LoWPAN Well Known Prefixes are carried inline in the packet. The bit field is aligned in little-endian format. Therefore the first byte's LSB refers to the first 6LoWPAN Well Known Prefix. In Remote Parameter Responses, all 6LoWPAN well known prefixes defined on the device will be included. In Remote Parameter Requests, this bit field defines which 6LoWPAN well known prefixes are defined. There must be a ***6LoWPAN Well Known Prefix*** field included for each bit that is set in the ***6LoWPAN Well Known Prefix Enable*** bit field.

6LoWPAN Well Known Prefix *N*

For each bit in the ***6LoWPAN Well Known Prefix Enable*** bit field, an 8-byte ***6LoWPAN Well Known Prefix*** field must be included in the packet. In Remote Parameter Set Packets, a value of $FFFFFFFFFFFFFFFF_{HEX}$ is used to unset a defined prefix. Any other value will set this value as a 6LoWPAN well known prefix.

Figure 3.9 IPv6 Address Encoding in Parameter Specifications



IPv6 Address Enable

This bit field determines which IPv6 addresses are carried inline in the packet. The bit field is aligned in little-endian format. Therefor the first byte's LSB refers to the first IPv6 address. In Remote Parameter Responses, all IPv6 addresses defined on the device will be included. In Remote Parameter Requests, this bit field defines which IPV6 addresses are defined. There must be a pair of ***IPv6 Address*** and ***IPv6 Address Status*** fields included for each bit that is set in the ***IPv6 Address Enable*** bit field.

IPv6 Address *N*

For each bit in the ***IPv6 Address Enable*** bit field, a 16-byte ***IPv6 Address*** field together with the corresponding ***IPv6 Address Status*** field must be included in the packet. For efficiency reasons, there are no measures implemented for even byte alignment; thus, each ***IPv6 Address*** field is aligned directly after the previous ***IPv6 Address Status*** field, resulting in a length of 17 bytes per included address.

IPv6 Address Status *N*

In Remote Parameter Responses, this one-byte field specifies the status of the addresses. In Remote Parameter Set Requests, this field is used to determine the operation to be performed. In order to configure or overwrite an address, use a value of 3. In order to delete an address, use a value of 4. All possible values are given in Table 3.3.

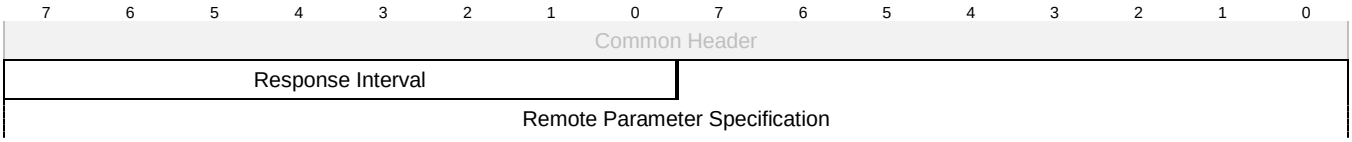
Table 3.3 Values for IPv6 Address Status Field

Value	Remote Parameter Response Packet	Remote Parameter Set Packet
1	Address is auto-configured (computed from prefix advertisements).	*MUST NOT* be used.
2	Address is DHCP configured (currently unused; DHCPv6 is not supported).	*MUST NOT* be used.
3	Address is manually configured (either assigned by the application, or link-local auto-configured address).	Define or overwrite an IPv6 address.
4	Not used.	Delete an IPv6 address definition.

3.4.2. Remote Parameter Request Packet

This packet type is used to query for the parameter configuration of the device. The parameters to be queried can be specified in the ***Remote Parameter Specification*** encoding.

Figure 3.10 Remote Parameter Request Packet Layout



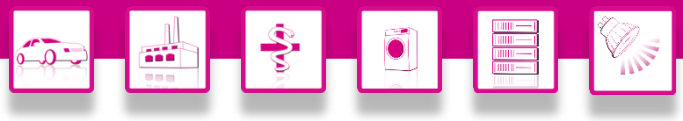
Response Interval

This field specifies a maximum number of seconds for the delay after which the receiver *MUST* send its Remote Parameter Response packet. The receiver *MUST* use a random number generator to compute the actual millisecond delay time of the response. The ***Response Interval*** field serves as an upper boundary for this interval.

The response interval *SHOULD* be set to a value larger than 0 for all Remote Parameter Requests that are sent to IPv6 multicast addresses. The optimal response time depends on the number of nodes in the network and on the selected modulation scheme.

Remote Parameter Specification

This variable length encoding defines which parameter values are requested from the target. The format of the ***Remote Parameter Specification*** encoding is defined in section 3.4. The ***Group Data*** encoding of the ***Remote Parameter Specification*** *MUST NOT* include a ***Parameter Value List****;* only Group and Parameter Selectors can be included.



3.4.2.1. Remote Parameter Request Packet Example

This section gives an example for a Remote Parameter Request packet intended to request the PAN ID, PAN Address, IPv6 Addresses, and configured modulation and radio channel. The request contains a Query ID and is targeted for devices only. The corresponding response is shown in section 3.4.3.1. For demonstration purposes, the request contains undefined Group and Parameter Specifiers.

Figure 3.11 Example Remote Parameter Request Packet

0	1	2	3	4	5	6	7	8	9	10
08_{HEX}	05_{HEX}	$2A_{HEX}$	xx_{HEX}	01_{HEX}	59_{HEX}	03_{HEX}	$0E_{HEX}$	82_{HEX}	02_{HEX}	45_{HEX}

Table 3.4 Remote Parameter Request Example Explanations

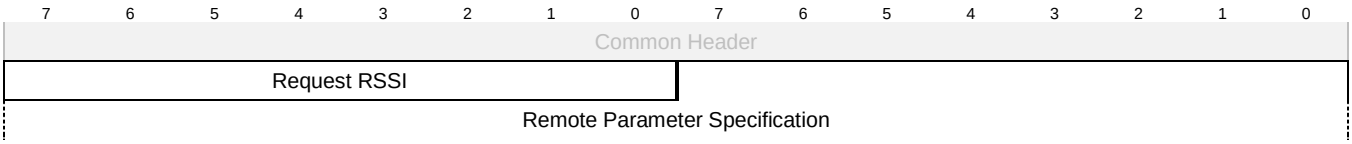
Byte	Fields	Description
Common Header		
0	Type = 08_{HEX}	This is a Remote Parameter Request packet.
	ARQ = 0	Receiver should not send an acknowledge.
1	Device = 1	Devices must respond to this packet.
	Bridge = 0	Gateways must not respond to this packet.
	QID = 1	There is a Query ID included in the packet.
	HCL = 0	No hop count limitation in place.
	PID = 0	No Product ID filter in place.
	VID = 0	No Vendor ID filter in place.
	OTAU = 0	No Over-the-Air Update filter in place.
2	Query ID = $2A_{HEX}$	This is the Query ID.
3	Hop Count Limit = xx_{HEX}	The value of the hop count limit does not matter as the corresponding flag is not set in byte 1.
Remote Parameter Request – Packet Specific Data		
4	Response Interval = 1	The receiver of this packet should send its response after 0 to 1 seconds.
5	Group Specifier = 59_{HEX}	Values from Generic (01_{HEX}), PHY Parameters (08_{HEX}) and Network Parameters Group (10_{HEX}) are requested. The additional bit (40_{HEX}) is currently unused but included to demonstrate the receiver behavior in such cases.
6	Generic Parameter Specifier = 03_{HEX}	PAN ID (01_{HEX}) and PAN Address (02_{HEX}) from the Generic Parameter Group are requested.
7	PHY Parameter Specifier = $0E_{HEX}$	Radio Channel (02_{HEX}) and Modulation (04_{HEX}) are requested from the PHY Parameter Group. The additional bit (08_{HEX}) is currently unused but included to demonstrate the receiver behavior in such cases.
8	NET Parameter Specifier = 82_{HEX}	Request Neighbor Cache Size (02_{HEX}) and indicate the availability of an extended Parameter Specifier (80_{HEX}).

Byte	Fields	Description
9	Extended NET Parameter Specifier = 02_{HEX}	Request the IPv6 Address Configuration (02_{HEX}).
10	Parameter Specifier for undefined group	This is the Parameter Specifier for the undefined group flagged in byte 5. Its meaning is not defined in the current NetMA configuration, but it is included here to demonstrate the receiver behavior in such cases.

3.4.3. Remote Parameter Response Packet

This packet type is sent by ZWIR45xx devices in response to Remote Parameter Requests. There is typically no need to generate such packets manually.

Figure 3.12 Remote Parameter Request Packet Layout



Request RSSI

This field contains the RSSI value of the Remote Parameter Request Packet that triggered this response. The value can be used for basic analysis of the link quality. However, note that this value only reports the link quality of the last hop of a link. For detailed link quality analysis, the Topology Detection feature (see section 2.2) or the Tracing (see section 2.3) feature *SHOULD* be used.

Remote Parameter Specification

This variable length encoding contains the data that has been requested via the packet triggering this response. If the request contained groups or data fields that are not supported by the responding device, the corresponding flags will be cleared and no data will be included for the respective parameters. This mechanism can be used to find out whether a node supports specific parameters or not.

3.4.3.1. Example

Figure 3.13 shows an example of a response to the request that was shown as an example for the Remote Parameter Request in section 3.4.2.1.

Figure 3.13 Example Remote Parameter Response Packet

0	1	2	3	4	5	6	7	8	9
89_{HEX}	41_{HEX}	$A7_{HEX}$	19_{HEX}	03_{HEX}	AC_{HEX}	CA_{HEX}	00_{HEX}	11_{HEX}	$7D_{HEX}$
00_{HEX}	00_{HEX}	$2F_{HEX}$	12_{HEX}	34_{HEX}	06_{HEX}	05_{HEX}	01_{HEX}	82_{HEX}	08_{HEX}
02_{HEX}	01_{HEX}	00_{HEX}	FE_{HEX}	80_{HEX}	00_{HEX}	00_{HEX}	00_{HEX}	00_{HEX}	00_{HEX}
00_{HEX}	02_{HEX}	11_{HEX}	$7D_{HEX}$	00_{HEX}	00_{HEX}	$2F_{HEX}$	12_{HEX}	34_{HEX}	03_{HEX}

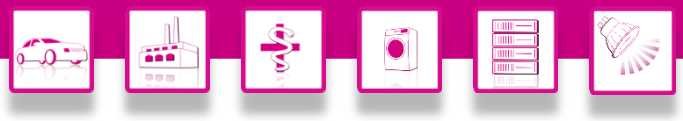


Table 3.5 Remote Parameter Response Example Explanations

Byte	Fields	Description
Common Header		
0	Type = 09_{HEX}	This is a Remote Parameter Response packet.
	ARQ = 1	Advise the receiver of this packet to send an acknowledge.
1	Device = 1	The responding module is configured in Device Mode (per request).
	OTAU = 1	The responding device is OTAU enabled.
Remote Parameter Response– Packet Specific Data		
2	RSSI = $A7_{HEX}$	The packet triggering this response was received with an RSSI value of -89dBm.
3	Group Specifier = 19_{HEX}	Values from Generic (01_{HEX}), PHY Parameters (08_{HEX}) and Network Parameters Group (10_{HEX}) are included in the response. The additional bit (40_{HEX}), which was set in the request, has been reset because the device does not support it. Consequently the packet elides the corresponding Parameter Specifier from the request as well.
4	Generic Parameter Specifier = 03_{HEX}	PAN ID (01_{HEX}) and PAN Address (02_{HEX}) from Generic Parameter Group are included in the parameter list.
5 – 6	PAN ID = $CAAC_{HEX}$	The PAN ID of the responding device.
7 – 14	MAC Address = 00:11:7D:00:00:2F:12:34	The MAC Address of the responding device.
15	PHY Parameter Specifier = 06_{HEX}	Radio Channel (02_{HEX}) and Modulation (04_{HEX}) from the PHY Parameter Group are included in the response. The additional bit (08_{HEX}) which was set in the request has been reset as the device does not support it.
16	Channel = 05_{HEX}	The device is using radio channel two.
17	Modulation = 01_{HEX}	The device is using QPSK modulation.
18	NET Parameter Specifier = 82_{HEX}	The response includes the Neighbor Cache Size (02_{HEX}) and signals the availability of an extended Parameter Specifier (80_{HEX}).
19	Neighbor Cache Size = 08_{HEX}	The device has a neighbor cache for up to eight items.
20	Extended NET Parameter Specifier = 02_{HEX}	The IPv6 Address Configuration is included in the response.
21 – 22	IPv6 Address Bit Field = 0001_{HEX}	There is one IPv6 Address (index 0) included in the response.
23 – 38	IPv6 Address 1 FE80::211:7D00:2F:1234	This is the IPv6 Address at index 0.
39	IPv6 Address 1 Status = 03_{HEX}	IPv6 Address at index 0 is a manually configured IPv6 Address.

3.4.4. Remote Parameter Set Packet

This type of packet requests the change of one or more parameters. Changes will be in effect temporarily until the next device reset or a parameter change triggered by the application firmware. To make parameter changes nonvolatile, use the Remote Parameter Store packet after changing the parameters.

Devices receiving Remote Parameter Set packets perform packet checking, verifying that all parameters included in the ***Remote Parameter Specification*** are supported and writable by the NetMA version running on the device. Furthermore, the packet length is matched against the length expected from the ***Parameter Group Specifiers*** and the ***Parameter Specifiers***. If one of these verification steps fails, a Reject packet is sent to the originator of the Remote Parameter Set Request and no changes are performed. The Reject packet is sent regardless of the value of the ARQ flag in the common header. Depending on the reason for rejection, Reject packets will carry one of the values defined in Table 3.6 in the ***Reject Reason*** field.

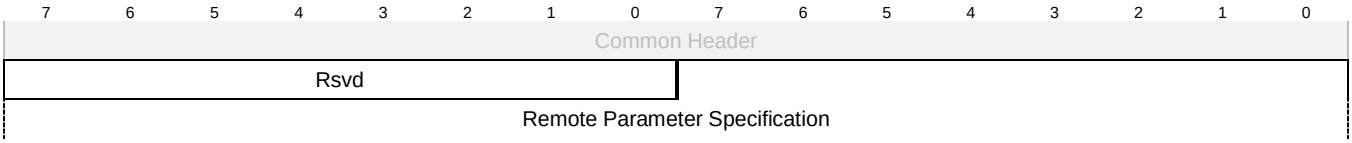
Table 3.6 Reject Reason Values

Value	Reject Reason
01_{HEX}	There is a mismatch between the packet length and the length expected from the ***Parameter Group Specifiers*** and the ***Parameter Specifiers***.
02_{HEX}	The ***Group Specifier*** of the ***Remote Parameter Specification*** has at least one unsupported Parameter Group enabled.
03_{HEX}	At least one of the ***Parameter Specifiers*** has enabled at least one unsupported parameter. This will happen for undefined parameters and for parameters that are read-only.

Receiving no Reject packet *SHOULD NOT* be considered as an indicator of successful completion of the Remote Parameter Set operation. Instead, setting the ARQ flag and waiting for the corresponding Acknowledge is *RECOMMENDED* for verifying successful completion of a Remote Parameter Set operation.

Note: Acknowledges on Remote Parameter Set packets are sent before the requested changes are performed. This ensures that the Acknowledge can be delivered even if network parameters are changed, which would make communication temporarily impossible; e.g., if the radio frequency is changed for the whole network.

Figure 3.14 Remote Parameter Set Packet Layout



Rsvd

This field is reserved for future use and *MUST* be set to 0.

Remote Parameter Specification

This encoding specifies the parameters together with the values to be set. The format of this field is given in section 3.4.

Important: Only writable parameters are allowed to be included in the ***Remote Parameter Specification***. If any read-only parameter or unknown parameter is included in the ***Remote Parameter Specification***, packet handling is rejected completely and a Reject Packet is sent, specifying the cause of the rejection.

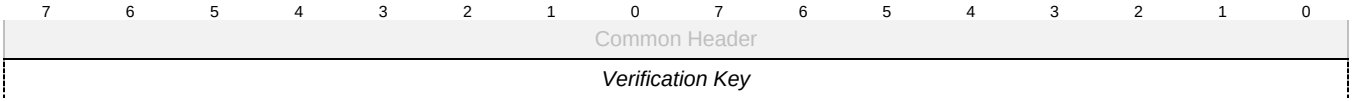
3.4.5. Remote Parameter Store Packet

This packet triggers the storage of the current parameter configuration to the device's flash memory.

Note: IPv6 addresses that are configured by DHCP or that are automatically configured are not stored to the flash.

Important: All parameters are stored regardless of the source of the parameter change. In order to prevent accidentally storing a parameter configuration, the packet requires a security key to be transmitted in the packet specific data.

Figure 3.15 Remote Parameter Store Packet Layout



Verfication Key

This field *MUST* carry the constant byte sequence 53_{HEX} 74_{HEX} $6F_{HEX}$ 72_{HEX} 65_{HEX} 21_{HEX}. If this field is missing or does not contain these specific bytes, a Reject Packet is sent carrying Reject Reason 04_{HEX}. The Reject packet is sent regardless of the status of the ARQ bit in the Remote Parameter Store packet.

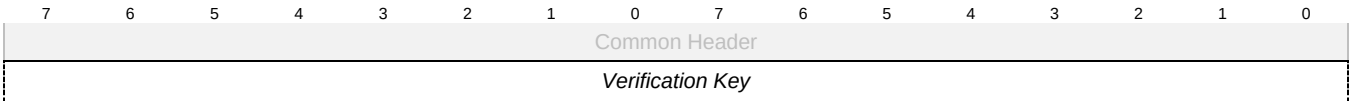
3.4.6. Remote Parameter Defaults Packet

This packet permanently restores all parameters to their factory default values. This task is accomplished by deletion of the flash page used for parameter storage. This request triggers a network reset of the receiving device.

If the ARQ bit is set in this request, the receiver will send the Acknowledge packet before restoring the factory default values.

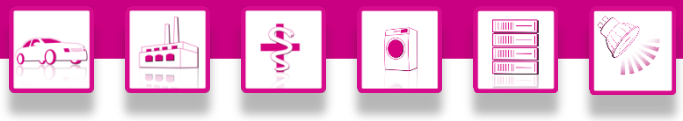
Important: This action cannot be undone. In order to prevent accidentally restoring the factory default settings, the packet requires a security key to be transmitted in the packet specific data.

Figure 3.16 Remote Parameter Defaults Packet Layout



Verification Key

This field *MUST* carry the constant byte sequence 44_{HEX} 65_{HEX} 66_{HEX} 61_{HEX} 75_{HEX} $6C_{HEX}$ 74_{HEX} 73_{HEX} 21_{HEX}. If this field is missing or does not contain these specific bytes, a Reject packet is sent carrying Reject Reason 04_{HEX}. The Reject packet is sent regardless of the status of the ARQ bit in the Remote Parameter Defaults Request.



3.5. Reset Device Packet

This packet triggers a system reset. This can be used to restore a parameter configuration that has been changed using a Remote Parameter Set Request. After the system reset, the last stored configuration will be in effect. If no configuration has been stored, the system default configuration will be active. There are no additional data included in this packet type. If the ARQ bit is set in this request, the receiver will send the Acknowledge packet before resetting the device.

3.6. Reset TRX Statistics Packet

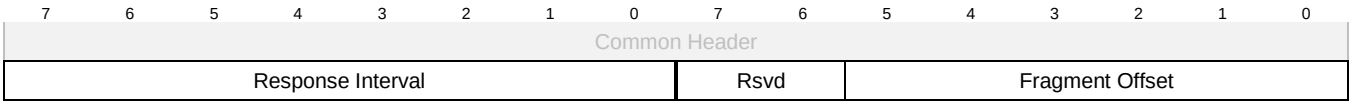
The reception of this packet triggers a reset of the TRX statistics. This can be used to define a starting point for duty-cycle monitoring or other traffic-monitoring tasks. There are no additional data included in this type of packet.

3.7. Routing Table Request Packet

Reception of this packet triggers the receiver to transmit its current routing table contents. This data provides information about which nodes routes exist at the time of the request. This feature is mainly useful for debugging purposes.

Responses will carry up to 32 routing table entries. Reading the whole routing table of devices with more than 32 routing table entries must be performed fragment-wise. In order to specify which routing table fragment should be returned, the Routing Table Request Packet provides a ***Fragment Offset*** field, specifying the starting offset as a multiple of 32. Reading the whole routing table is accomplished by subsequent Routing Table Requests with the ***Fragment Offset*** field increased in each packet.

Figure 3.17 Routing Table Request Packet Layout

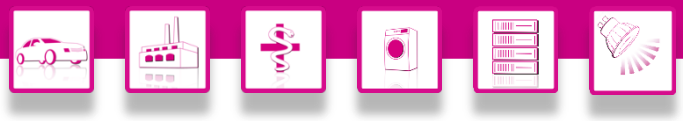


Response Interval

This field specifies the maximum time in seconds after which the Routing Table Response is sent to the target node. The time is chosen randomly from the interval 0 < (1000 ∗ ***Response Interval***) in milliseconds. If the Routing Table Request is sent to an unicast address, this time can be set to 0. If the request is sent to a multicast address, a time appropriate for the network size and modulation scheme *SHOULD* be used to maximize the probability of successful transmission of the result.

Rsvd

These bits are reserved for future use and *MUST* be set to 0.



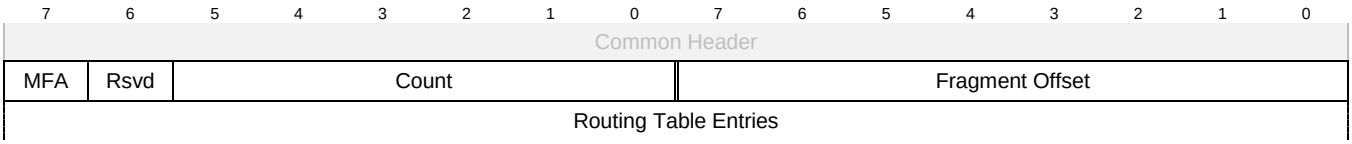
Fragment Offset

This field specifies the offset of routing table entries if more than 32 entries are stored in the routing table. The first request *SHOULD* always be sent with a Fragment Offset of 0. If the response will indicate the availability of additional entries, the Fragment Offset *SHOULD* be incremented by one in consecutive requests to obtain the full set of routing table entries. A response to a Routing Table Request will include the routing table entries (***Fragment Offset*** ∗ 32) ≤ N < ((***Fragment Offset*** + 1) ∗ 32). If fewer routing table entries than specified by the Fragment Offset exist, the corresponding Routing Table Response (section 3.8) will be empty with the ***MFA*** flag set to 0.

3.8. Routing Table Response Packet

Routing Table Response packets are sent as the answer to Routing Table Requests. A response can carry up to 32 routing table entries.

Figure 3.18 Routing Table Response Packet Layout



MFA

This flag indicates that there are more routing table fragments available that can be requested in subsequent Routing Table Requests.

Rsvd

Reserved for future use.

Count

This field specifies how many routing table entries are included in the packet. The number range is 0 to 32.

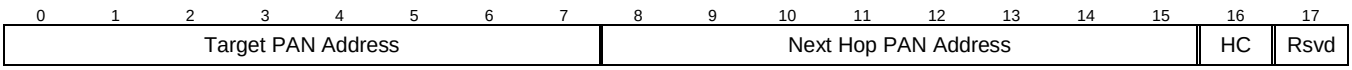
Fragment Offset

This field is a copy of the ***Fragment Offset*** field of the Routing Table Request that activated the response.

Routing Table Entries

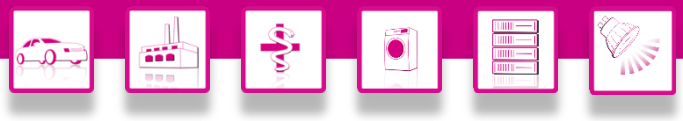
This variable length field contains *Count* routing table entries, which are encoded in the format specified in Figure 3.19.

Figure 3.19 Routing Table Entries Encoding



Target PAN Address

This is the 8-byte route target PAN address.



Next Hop PAN Address

This is the 8-byte next hop PAN address that must be taken to reach the route target.

HC

This is the number of remaining hops to the route target.

Rsvd

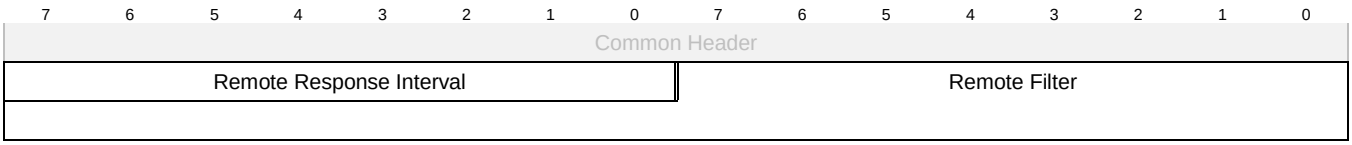
This field is reserved for future use.

3.9. Neighbor Request Packet

Neighbor requests are intended to find the neighbor devices of the device being queried. The information returned on Neighbor Requests includes the PAN address of the neighboring node together with bi-directional link quality information. Neighbor Requests can be used to determine network topology or to find neighbors with specific parameters.

A node receiving a Neighbor Request packet sends out a Remote Parameter Request asking for its neighbors' PAN Address as a broadcast message. An internal Remote Parameter Response handler is activated that collects the returned neighbor information. The Remote Parameter Request includes filter flags as specified in the ***Remote Filter*** field of the Neighbor Request. Using these filters, the scope of the requested neighbors can be limited or multiple neighbor requests can be sent using the same Query ID to ensure that all neighbors respond successfully. The Remote Parameter Request uses the timeout value specified in the Neighbor Request Packet as its Response Interval. After sending the Remote Parameter Request, the NetMA2 protocol stack waits for one second longer than the specified timeout to collect Remote Parameter Responses. After this time, a Neighbor Response Packet is sent to the sender of the Neighbor Request Packet, containing the neighbor information of all neighbors that have answered the Remote Parameter Request.

Figure 3.20 Neighbor Request Packet Layout



Remote Response Interval

This field specifies the response interval that is used for the internal Remote Parameter Request. The Neighbor Response Packet is sent after (***Remote Response Interval*** + 1 second).

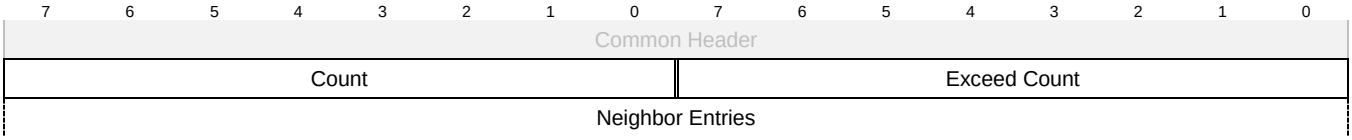
Remote Filter

This variable length encoding contains the filter specification that is included in the Remote Parameter Request. The filter encoding is the same as specified in section 3.1.2 for the common header. The length of this encoding is at least one byte.

3.10. Neighbor Response Packet

Neighbor Responses are sent by devices as the answer to Neighbor Requests. They contain information about neighbors that are directly reachable from the responding device. In the rare case of more than 100 directly reachable nodes, only the first 100 responses will be included together with the ***Exceed Count*** field set to the number of nodes from which responses have been received but were not included in the packet. If there is a probability of having more than 100 nodes in the direct range of a node, the ***Remote Filter*** field in the Remote Parameter Request *SHOULD* include a Query ID filter and the Neighbor Request *SHOULD* be repeated with the same Query ID filter until the ***Exceed Count*** field of the corresponding response is 0.

Figure 3.21 Neighbor Response Packet Layout



Count

This field specifies how many neighbor entries are included in the response. The maximum number of neighbors included in a response is 100.

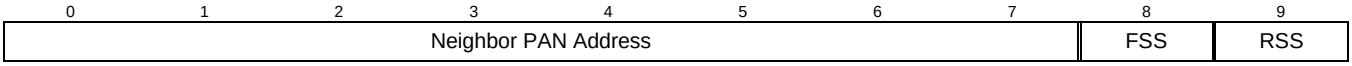
Exceed Count

This field specifies the number of neighbors that are present for the current node but their answers have not been considered due to the neighbor response size limitation of 100 responding nodes.

Neighbor Entries

This variable length encoding contains *Count* Neighbor Entries, which have the format specified Figure 3.22.

Figure 3.22 Format of the Neighbor Entries Encoding



Neighbor PAN Address

This is the PAN address of the neighbor.

FSS

Forward Signal Strength: This field specifies the RSSI value at which the Remote Parameter Request has been received.

RSS

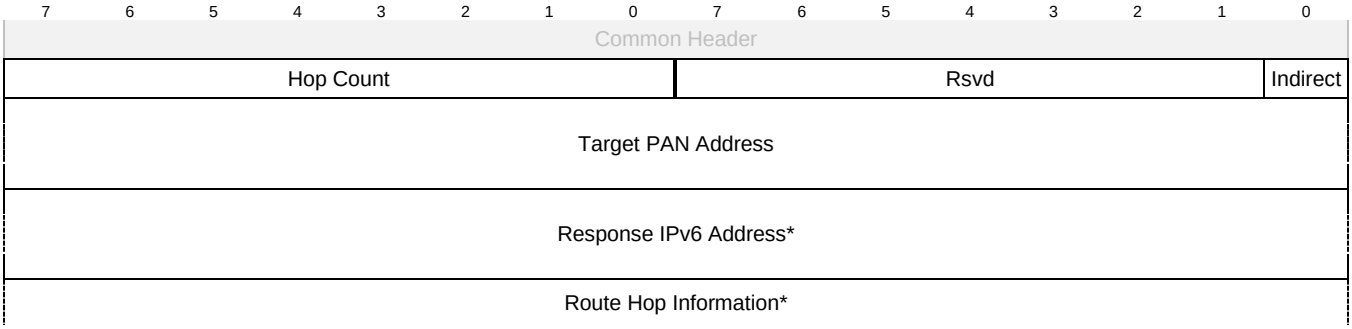
Return Signal Strength: This field specifies the RSSI value at which the Remote Parameter Response has been received.

3.11. Trace Request Packet

NetMA2's tracing functionality allows examination of routes and their link quality. Tracing can be started from the sender of the Trace Request itself if it is a ZWIR45xx module or from arbitrary nodes having the ***Indirect*** field set to 1. For indirect Trace Requests, the packet must be sent to the intended starting node of the trace.

During the execution of the Trace Request, the NetMA2 stack examines the routing table of the nodes along the route to determine the next hop. Each hop increments the ***Hop Count*** field and appends its own PAN address along with the RSSI value of the received Trace Request and forwards the extended request to the next hop. This procedure is executed until the ***Target PAN Address*** field matches the PAN address of the receiving node. If this match is detected, the node sends the Trace Response to the address specified in the ***Response IPv6 Address*** field. If one of the nodes on the route does not have an entry matching the ***Target PAN Address*** field in its routing table, a Trace Fail packet is sent to the address specified in the ***Response IPv6 Address*** field. This response contains the route information up to the failing node.

Figure 3.23 Trace Request Packet Layout



Hop Count

This field specifies how many ***Route Hop Information*** encodings are included in the request.

Rsvd

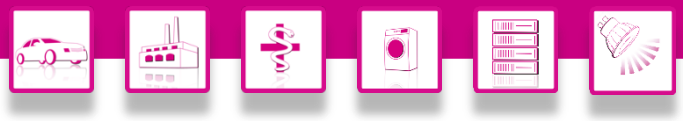
This field is reserved for future use and *MUST* be set to 0.

Indirect

This flag specifies that the trace should be started from the node receiving this request. The node receiving this request will set the ***Hop-Count*** field to zero and set the ***Response IPv6 Address*** field to the IPv6 address of the sender of this packet. Before forwarding the request, the flag will be cleared. Note that for indirect Trace Requests, no ***Response IPv6 Address*** field needs to be included in the request packet. If there is a ***Response IPv6 Address*** field included, this field will be overwritten with the packet sender's IPv6 address.

Target PAN Address

This is the PAN address of the route's target node.



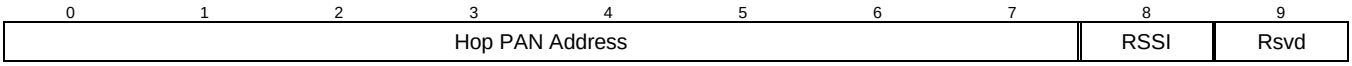
Response IPv6 Address

This is the IPv6 address of the node that should receive the trace response. For direct tracing, this field *SHOULD* contain the address of the originator of the Trace Request. If the ***Indirect*** flag is set, this field can be omitted. If it is present in indirect tracing, the field will be overwritten.

Route Hop Information

This encoding contains a list of *Hop Count* elements, each one having the format specified in Figure 3.24. Each item contains the PAN Address of the hop and the corresponding RSSI value with which it was reached from the previous hop.

Figure 3.24 Layout of the Route Hop Information Field



Hop PAN Address

This is the hop's PAN address.

RSSI

This is the RSSI value at which the Trace Request Packet was received by the node specified in Hop PAN Address.

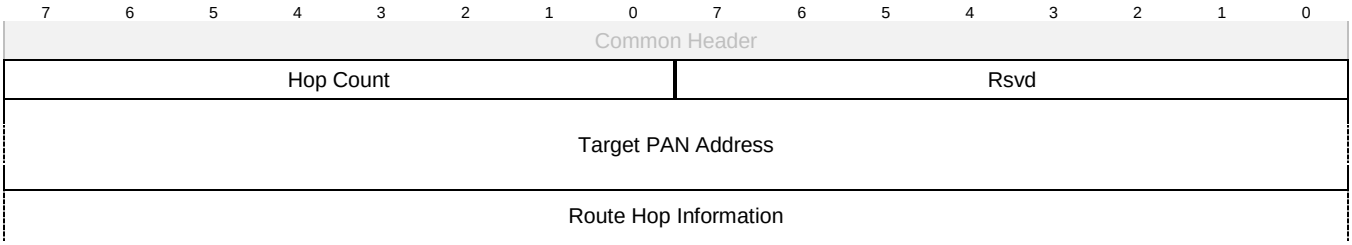
Rsvd

Reserved for future use.

3.12. Trace Fail and Trace Response Packet

One of these two packet types will be received as the answer to a Trace Request Packet.

Figure 3.25 Trace Fail and Trace Response Packet Layout



Hop Count

This field specifies how many ***Route Hop Information*** encodings are included in the packet. Note that Trace Fail packets can also include hop information.

Rsvd

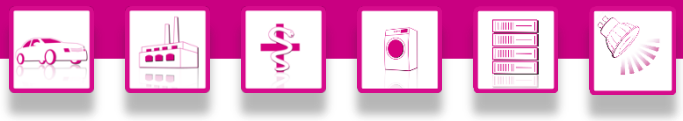
Reserved for future use.

Target PAN Address

This is the PAN address of the route's target node.

Route Hop Information

This encoding contains a list of *Hop Count* elements. Each item contains the PAN Address of the hop and the corresponding RSSI value with which it was reached from the previous hop. For the format specification of the ***Route Hop Information*** encoding, refer to section 3.11.



4 Using NetMA Directly on ZWIR45xx Nodes

NetMA has been designed to provide debugging functionality for devices such as personal computers that are not directly incorporated in the wireless device network. For that reason, the ZWIR45xx NetMA implementation does not provide handler functions for NetMA responses. NetMA responses received by a ZWIR45xx device are ignored by default. Furthermore, there are no API functions providing high-level access to NetMA functionality. However, in some cases it might be desirable to use the NetMA functionality directly from ZWIR45xx devices.

In order to make use of NetMA functionality, nodes must generate NetMA requests explicitly, sending them to the NetMA UDP port, using the functions **`ZWIR_SendUDP2`** (refer to the *ZWIR451x Programming Guide* for a detailed reference on these functions). Note that the function **`ZWIR_SendUDP`** cannot be used for sending NetMA packets, as this would require opening a duplicate socket, which would result in an error. In order to receive NetMA packets in the application, the function **`ZWIR_NetMA_ResponseHandler`** must be defined in the application code. Refer to the NetMA API reference below for detailed information about the available functions.

4.1. NetMA2 API Reference

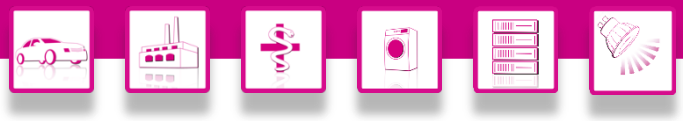
```
void
  ZWIR_NetMA_SetPort ( uint16_t  port );
```

This function sets the UDP port used by the NetMA protocol to the value of ***`port`***. The default value of the NetMA port is 61356.

```
bool
  ZWIR_NetMA_ResponseHandler ( uint8_t  const*  data,
                               uint16_t         size );
```

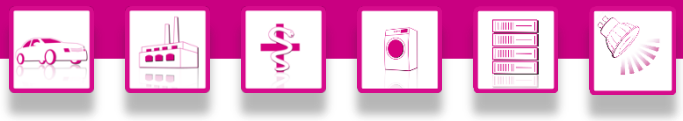
In order to receive NetMA packets in the application, this hook must be implemented. If no implementation is provided, only NetMA requests are processed as usual, but no responses are received. If the application sends NetMA requests into the network, this function must be provided to receive the corresponding responses. The ***`data`*** argument contains a pointer to the UDP payload of the packet, pointing to the first byte of the NetMA Common Header. The ***`size`*** argument contains the packet length in bytes. The return value of the function determines whether the packet has been handled by the application code or not. If true is returned, the packet has been handled and no further action will be taken by the NetMA stack. If false is returned, the NetMA stack considers this packet as unhandled and will try to execute its internal handlers, providing the default behavior.

Note: This function is called for each incoming NetMA packet. Thus, if an external device sends a NetMA request to the all node's multicast addresses or to this particular device, this hook will be called and the packet must be handled.



```
bool
  ZWIR_NetMA_Filter ( uint8_t*  data,
                      uint16_t  size,
                      uint8_t*  dataOffset );
```

This function is provided for convenience, applying the NetMA filtering rules. The ***data*** argument is the pointer to the packet, the ***size*** argument is the packet size, and the ***dataOffset*** argument is a pointer to a data offset value that is written by the filtering function. The function returns ***true*** if the packet must be dropped and ***false*** if packet processing should be continued. After completion of this function, the value pointed to by ***dataOffset*** carries the byte offset of the NetMA packet-specific data.



5 Security Considerations

Including libZWIR45xx-NetMA2.a provides deep insight into the network configuration. Furthermore, including libZWIR45xx-NetMA2-Ext.a adds the possibility of changing all network settings from any remote host. Application developers must be aware of the risk of external attackers using this functionality to impair the application behavior. Therefore, applications *SHOULD* use security mechanisms to protect themselves from malicious modification of the network settings. Including libZWIR45xx-NetMA2-Ext.a into production firmware without securing the communication will expose users to a significant risk of being a target of a Denial of Service (DoS) attack through NetMA.

ZMDI's network stack for ZWIR45xx devices provides security by means of Internet Protocol Security (IPSec) and Internet Key Exchange version 2 (IKEv2), which are both well-known and widely used protocols. All modern operating systems can handle these protocols, so there is no reason not to use them. The sections below show how IPSec *SHOULD* be configured in conjunction with NetMA to provide the full functionality without security flaws. For a full reference of ZMDI's IPSec and IKEv2 implementation, refer to the *ZWIR45xx Application Note – Guide for Using IPSec and IKEv2 in 6LoWPANs.*

5.1. Common IPSec Configuration

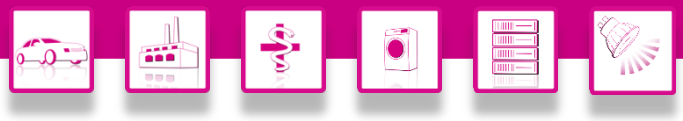
For most application scenarios, it is desirable to use common IPSec keys for all nodes. Using common keys has the advantage that NetMA can be used without restrictions. The common key can be defined solely for NetMA or it can be shared with the application. Instructions for distributing the common key are not within the scope of this document.

In order to set up secure communication, the corresponding security policy and the security association must be defined by the application(s) running on the wireless devices and any computer that wishes to use the NetMA functionality of the network. This section demonstrates the configuration of the device, leaving the computer configuration to the user. See the *ZWIR45xx Application Note – Guide for Using IPSec and IKEv2 in 6LoWPANs* for a computer configuration example.

The first step is defining the common security association. The following example assumes hard-coded keys.

```
ZWIRSEC_EncryptionSuite_t  e = { ZWIRSEC_encAESCTR, "A16CharCryptKey!", "1234" };

ZWIRSEC_AuthenticationAlgorithm_t a = { ZWIRSEC_autAESXCBC96, "A16CharAuthKey! " };

ZWIRSEC_SecurityAssociation* sa = ZWIRSEC_AddSecurityAssociation ( 12345, &e, &a );
```

In this example, the first two statements define the encryption and authentication methods, along with the corresponding key data. It is important to use the same values on all devices being configured. The third line adds the security association to the Security Association Database, making the data available to the IPSec network stack for use with security policies.



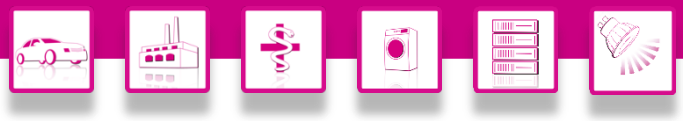
The policies required for the use with NetMA only are defined below:

```
ZWIR_IPv6Address_t  addr = IPV6_UNSPECIFIED;

ZWIRSEC_AddSecurityPolicy ( ZWIRSEC_ptOutputApply, &addr, 0,
                            ZWIR_protoUDP, 61356, 61356, sa);

ZWIRSEC_AddSecurityPolicy ( ZWIRSEC_ptInputApply, &addr, 0,
                            ZWIR_protoUDP, 61356, 61356, sa );
```

Here, the first line defines an address to be used in the security policies. The first policy being created advises the IPSec stack to apply the security association defined above (***sa***) to be used on UDP packets (***ZWIR_protoUDP***) on port ***61356*** sent (***ZWIRSEC_ptOutputApply***) to any address (***addr, 0***). The second policy does the same for incoming traffic (***ZWIRSEC_ptInputApply***). The code above must be executed on all devices in the network.

5.2. Dedicated IPSec Configuration

In some cases, the application developer might decide to have dedicated security keys for each pair of communicating nodes. This can be performed manually or by means of IKEv2. This level of security can be used with NetMA, but the NetMA functionality will be reduced significantly. It will no longer be possible to use IPv6 multicast requests; unicast requests must be used instead. As a consequence, the Neighbor Request and the Trace Request packets will not work in such configurations.



6 Related Documents

Note: X_xy refers to the current revision of the document.

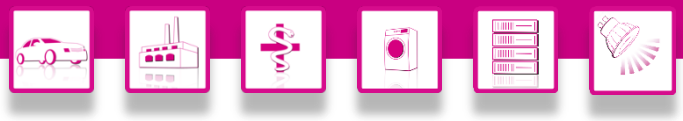
Document	File Name
*ZWIR451x Programming Guide**	*ZWIR451x_ProgGuide_revX_xy.pdf*
*ZWIR45xx Application Note – Guide for Using IPSec and IKEv2 in 6LoWPANs**	*ZWIR45xx_AN_Security_revX_xy.pdf*

Visit the ZWIR4512 product page www.zmdi.com/zwir4512 on ZMDI's website www.zmdi.com or contact your nearest sales office for the latest version of these documents.

* Documents marked with an asterisk require a free customer login account. To set up a login account, click on Login in the upper right corner of the web page and follow the instructions.

7 Glossary

Term	Description
6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
API	Application Programming Interface
BPSK	Binary Phase-Shift Keying
DHCP	Dynamic Host Configuration Protocol
FSS	Forward Signal Strength
IKEv2	Internet Key Exchange version 2
IPSec	Internet Protocol Security
IPv6	Internet Protocol Version 6
LSB	Least Significant Bit
MSB	Most Significant Bit
NetMA	Network Monitoring and Administration Protocol
PAN	Personal Area Network
QPSK	Quadrature Phase Shift Keying
RSS	Return Signal Strength
RSSI	Received Signal Strength Indication
SA	Security Association
SAD	Security Association Database
SP	Security Policy
SPD	Security Policy Database
UDP	User Datagram Protocol



8 Document Revision History

Revision	Date	Description
1.00	August 24, 2014	First release.

Sales and Further Information

www.zmdi.com wpan@zmdi.com

Zentrum Mikroelektronik Dresden AG
Global Headquarters
Grenzstrasse 28
01109 Dresden, Germany

Central Office:
Phone +49.351.8822.306
Fax +49.351.8822.337

ZMD America, Inc.
1525 McCarthy Blvd., #212
Milpitas, CA 95035-7453
USA

USA Phone 1.855.275.9634
Phone +1.408.883.6310
Fax +1.408.883.6358

Zentrum Mikroelektronik Dresden AG, Japan Office
2nd Floor, Shinbashi Tokyu Bldg.
4-21-3, Shinbashi, Minato-ku
Tokyo, 105-0004
Japan

Phone +81.3.6895.7410
Fax +81.3.6895.7301

ZMD FAR EAST, Ltd.
3F, No. 51, Sec. 2,
Keelung Road
11052 Taipei
Taiwan

Phone +886.2.2377.8189
Fax +886.2.2377.8199

Zentrum Mikroelektronik Dresden AG, Korea Office
U-space 1 Building
11th Floor, Unit JA-1102
670 Sampyeong-dong
Bundang-gu, Seongnam-si
Gyeonggi-do, 463-400
Korea

Phone +82.31.950.7679
Fax +82.504.841.3026

European Technical Support
Phone +49.351.8822.7.772
Fax +49.351.8822.87.772

European Sales (Stuttgart)
Phone +49.711.674517.55
Fax +49.711.674517.87955