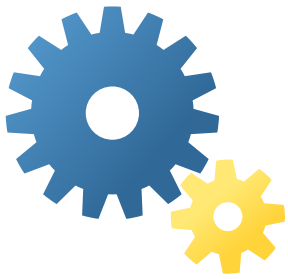

DHCPKit

DHCPKit Documentation

Release 1.0.1

S.J.M. Steffann

Apr 29, 2021

Contents

This DHCPKit extension allows DHCPKit to be used as the DHCPv6 server of a Vector Packet Processing (VPP) router. See https://fd.io/technology and https://wiki.fd.io/view/VPP for more information on VPP.

The official documentation[1] is hosted by Read the Docs[2].

[1] http://dhcpkit.readthedocs.io
[2] https://readthedocs.org

CHAPTER 1

Documentation

1.1 IPv6 Server extension configuration

This is the documentation of the configuration options of the dhcpkit_vpp package.

1.1.1 Overview of sections

Vpp-interface

VPP Interface to listen on

Example

```
<vpp-interface GigabitEthernet0/1/2>
    reply-from fe80::1
    link-address 2001:db8::1
</vpp-interface>
```

Section parameters

accept-multicast Whether to process multicast messages received on this interface

Default: “yes”

accept-unicast Whether to process unicast messages received on this interface

Default: “yes”

reply-from The link-local address to send on-link replies from

Default: The first link-local address found on the interface

link-address A global unicast address used to identify the link to filters and handlers. It doesn’t even need to exist.

Default: The first global unicast address found on the interface, or `::` otherwise

1.1.2 Overview of section types

Listeners

Configuration sections that define listeners. These are usually the network interfaces that a DHCPv6 server listens on, like the well-known multicast address on an interface, or a unicast address where a DHCPv6 relay can send its requests to.

Listen-vpp

This listener sets up a two-way connection to a VPP instance using Unix domain sockets. It will learn the server created by the VPP instance using the VPP Python API. The name of the socket endpoint it creates for itself (so VPP can send messages to DHCPKit) is specified as the name of the section.

With this listener DHCPKit can become a DHCPv6 server for VPP. You must list all VPP interfaces that DHCPKit should respond to.

VPP must be configured to create a punt socket:

```
punt {
    socket /run/vpp/punt_socket
}
```

This socket is used to send messages from DHCPKit back to VPP. You don't need to specify this socket in the DHCPKit configuration, it will learn it through the VPP API.

Example

```
<listen-vpp /run/vpp/client_socket>
    namespace-prefix foo

    <vpp-interface tap-0 />

    <vpp-interface GigabitEthernet0/1/2>
        reply-from fe80::1
        link-address 2001:db8::1
    </vpp-interface>
</listen-vpp>
```

Section parameters

mark (multiple allowed) Every incoming request can be marked with different tags. That way you can handle messages differently based on i.e. which listener they came in on. Every listener can set one or more marks. Also see the marked-with filter.

Default: "unmarked"

namespace-prefix Namespace prefix for the API. When specifying a prefix in the VPP startup configuration:

```
api-segment {
    prefix foo
}
```

then specify *foo* here.

Example: "namespace-prefix foo"

api-definitions Path to the JSON files that define the API. If left empty the default path for your system will be used.

Example: "api-definitions /usr/share/vpp/api"

Possible sub-section types

Vpp-interface **(page 3) (required, multiple allowed)** VPP Interface to listen on

1.2 dhcpkit_vpp package

1.2.1 Subpackages

dhcpkit_vpp.listeners package

Subpackages

dhcpkit_vpp.listeners.vpp package

Factory for the implementation of a listener on a VPP punt socket

exception dhcpkit_vpp.listeners.vpp.**UnknownVPPAction**
Bases: *dhcpkit_vpp.listeners.vpp.UnwantedVPPMessage* (page 5)

Signal that this message is incomplete because it contained an unknown VPP action value.

exception dhcpkit_vpp.listeners.vpp.**UnknownVPPInterface**
Bases: *dhcpkit_vpp.listeners.vpp.UnwantedVPPMessage* (page 5)

Signal that this message is incomplete because it came from an unknown VPP interface.

exception dhcpkit_vpp.listeners.vpp.**UnwantedVPPMessage**
Bases: dhcpkit.ipv6.server.listeners.IgnoreMessage

This is a message that we don't want

Submodules

dhcpkit_vpp.listeners.vpp.config module

dhcpkit_vpp.listeners.vpp.vpp module

dhcpkit_vpp.listeners.vpp.vpp_interface module

dhcpkit_vpp.protocols package

Classes and constants for protocol implementations

class dhcpkit_vpp.protocols.**Layer2Frame**
Bases: dhcpkit.protocol_element.ProtocolElement

Base class for layer 2 frames

class dhcpkit_vpp.protocols.**Layer3Packet**
Bases: dhcpkit.protocol_element.ProtocolElement

Base class for layer 3 packets

get_pseudo_header(*for_payload: dhcpkit_vpp.protocols.Layer4Protocol*) → bytes
Return the pseudo header for this protocol

Parameters **for_payload** – Get the pseudo header for the given layer 4 protocol

Returns The pseudo header bytes

class dhcpkit_vpp.protocols.**Layer4Protocol**
Bases: dhcpkit.protocol_element.ProtocolElement

Base class for layer 4 protocols

length
Return the length of this protocol+payload

Returns The length

protocol_number = 0

save(*zero_checksum: bool = False*, *recalculate_checksum_for: Optional[dhcpkit_vpp.protocols.Layer3Packet] = None*) → bytearray
Save the internal state of this object as a buffer.

Parameters

- **zero_checksum** – Save with zeroes where the checksum should be
- **recalculate_checksum_for** – Recalculate the checksum for the given layer 3 packet headers

Returns The buffer with the data from this element

Submodules

dhcpkit_vpp.protocols.layer2 module

Classes and constants for layer 2 frames

class dhcpkit_vpp.protocols.layer2.**Ethernet**(*destination: bytes = b'x00x00x00x00x00x00'*, *source: bytes = b'x00x00x00x00x00x00'*, *ethertype: int = 0*, *payload: dhcpkit.protocol_element.ProtocolElement = None*)
Bases: *dhcpkit_vpp.protocols.Layer2Frame* (page 5)

The class for ethernet frames.

classmethod determine_class(*buffer: bytes*, *offset: int = 0*) → type
Return the appropriate class to parse this element with.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

Returns The best known class for this data

display_destination() → dhcpkit.protocol_element.ElementDataRepresentation
Nicer representation of destination :return: Representation of destination

display_ethertype() → dhcpkit.protocol_element.ElementDataRepresentation
Nicer representation of ethertype :return: Representation of ethertype

display_source() → dhcpkit.protocol_element.ElementDataRepresentation
Nicer representation of source :return: Representation of source

load_from(*buffer: bytes*, *offset: int = 0*, *length: int = None*) → int
Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

Returns The number of bytes used from the buffer

save() → bytes
Save the internal state of this object as a buffer.

Returns The buffer with the data from this element

validate()
Validate that the contents of this object conform to protocol specs.

dhcpkit_vpp.protocols.layer3 module

Classes and constants for layer 3 protocols

class dhcpkit_vpp.protocols.layer3.**IPv6**(*traffic_class: int = 0*, *flow_label: int = 0*, *next_header: int = 0*, *hop_limit: int = 0*, *source: ipaddress.IPv6Address = None*, *destination: ipaddress.IPv6Address = None*, *payload: dhcpkit.protocol_element.ProtocolElement = None*)
Bases: *dhcpkit_vpp.protocols.Layer3Packet* (page 5)

The class for IPv6 packets.

classmethod determine_class(*buffer: bytes*, *offset: int = 0*) → type
Return the appropriate class to parse this element with.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

Returns The best known class for this data

get_pseudo_header(*l4_payload: dhcpkit_vpp.protocols.Layer4Protocol*) → bytes
Return the pseudo header for this protocol

Parameters **l4_payload** – The payload protocol to calculate the pseudo header for

Returns The pseudo header bytes

load_from(*buffer: bytes*, *offset: int = 0*, *length: int = None*) → int
Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

Returns The number of bytes used from the buffer

save() → bytearray
Save the internal state of this object as a buffer.

Returns The buffer with the data from this element

validate()
Validate that the contents of this object conform to protocol specs.

class dhcpkit_vpp.protocols.layer3.**UnknownLayer3Packet**(*data: bytes = b''*)
Bases: *dhcpkit_vpp.protocols.Layer3Packet* (page 5), dhcpkit.protocol_element.UnknownProtocolElement

A layer 3 packet of unknown type

classmethod determine_class(*buffer: bytes*, *offset: int = 0*) → type
Return the appropriate class to parse this element with.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

Returns The best known class for this data

get_pseudo_header(*for_payload: dhcpkit_vpp.protocols.Layer4Protocol*) → bytes
We don't have a pseudo header

Parameters **for_payload** – Get the pseudo header for the given layer 4 protocol

Returns The pseudo header

dhcpkit_vpp.protocols.layer3_registry module

The protocol layer 3 registry

class dhcpkit_vpp.protocols.layer3_registry.**ProtocolLayer3Registry**
Bases: dhcpkit.registry.Registry

Registry for Protocols

entry_point = 'dhcpkit_vpp.protocols.layer3'

dhcpkit_vpp.protocols.layer4 module

Classes and constants for layer 4 protocols

class dhcpkit_vpp.protocols.layer4.**UDP**(*source_port: int = 0*, *destination_port: int = 0*, *checksum: int = 0*, *payload: bytes = b''*)
Bases: *dhcpkit_vpp.protocols.Layer4Protocol* (page 6)

The class for UDP packets.

calculate_checksum(*l3_packet: dhcpkit_vpp.protocols.Layer3Packet*)
Calculate the checksum based on the current payload and the provided layer 3 packet.

Parameters **l3_packet** – The layer 3 packet that contains this UDP message

Returns The calculated checksum

classmethod determine_class(*buffer: bytes*, *offset: int = 0*) → type
Return the appropriate class to parse this element with.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

Returns The best known class for this data

length
Return the length of this protocol+payload

Returns The length

load_from(*buffer: bytes*, *offset: int = 0*, *length: int = None*) → int
Load the internal state of this object from the given buffer. The buffer may contain more data after the structured element is parsed. This data is ignored.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading
- **length** – The amount of data we are allowed to read from the buffer

Returns The number of bytes used from the buffer

protocol_number = 17

save(*zero_checksum: bool = False*, *recalculate_checksum_for: dhcpkit_vpp.protocols.Layer3Packet = None*) → bytearray
Save the internal state of this object as a buffer.

Parameters

- **zero_checksum** – Save with zeroes where the checksum should be
- **recalculate_checksum_for** – Recalculate the checksum for the given layer 3 packet headers

Returns The buffer with the data from this element

validate()
Validate that the contents of this object conform to protocol specs.

class dhcpkit_vpp.protocols.layer4.**UnknownLayer4Protocol**(*data: bytes = b''*)
Bases: *dhcpkit_vpp.protocols.Layer4Protocol* (page 6), dhcpkit.protocol_element.UnknownProtocolElement

A layer 3 packet of unknown type

classmethod determine_class(*buffer: bytes*, *offset: int = 0*) → type
Return the appropriate class to parse this element with.

Parameters

- **buffer** – The buffer to read data from
- **offset** – The offset in the buffer where to start reading

Returns The best known class for this data

length
The length of our data

Returns The length

save(*zero_checksum: bool = False*, *recalculate_checksum_for: dhcpkit_vpp.protocols.Layer3Packet = None*) → bytearray
Save the internal state of this object as a buffer.

Parameters

- **zero_checksum** – Save with zeroes where the checksum should be
- **recalculate_checksum_for** – Recalculate the checksum for the given layer 3 packet headers

Returns The buffer with the data from this element

dhcpkit_vpp.protocols.layer4_registry module

The protocol layer 4 registry

class dhcpkit_vpp.protocols.layer4_registry.**ProtocolLayer4Registry**
Bases: dhcpkit.registry.Registry

Registry for Protocols

entry_point = 'dhcpkit_vpp.protocols.layer4'

dhcpkit_vpp.protocols.utils module

dhcpkit_vpp.protocols.utils.**ones_complement_checksum**(*msg: Union[bytes, bytearray]*)

Calculate the 16-bit one's complement of the one's complement sum of a message.

Parameters **msg** – The message

Returns The checksum

dhcpkit_vpp.tests package

All the unit tests go here

Subpackages

dhcpkit_vpp.tests.protocols package

class dhcpkit_vpp.tests.protocols.**FrameTestCase**(*methodName='runTest'*)
Bases: unittest.case.TestCase

check_unsigned_integer_property(*property_name: str*, *size: int = None*)

Perform basic verification of validation of an unsigned integer

Parameters

- **property_name** – The property under test
- **size** – The number of bits of this integer field

parse_packet()

setUp()

Hook method for setting up the test fixture before exercising it.

test_class()

test_length()

test_parse()

test_save_fixture()

test_save_parsed()

test_validate()

Submodules

dhcpkit_vpp.tests.protocols.test_base_classes module

class dhcpkit_vpp.tests.protocols.test_base_classes.**Layer3PacketTestCase**(*methodName='runTes*
Bases: unittest.case.TestCase

test_abstract_get_pseudo_header()

class dhcpkit_vpp.tests.protocols.test_base_classes.**Layer4ProtocolTestCase**(*methodName='run*
Bases: unittest.case.TestCase

test_abstract_length()

test_abstract_save()

dhcpkit_vpp.tests.protocols.test_layer2 module

Test whether layer 2 parsing and generating works

class dhcpkit_vpp.tests.protocols.test_layer2.**Layer2FrameTestCase**(*methodName='runTest'*)
Bases: *dhcpkit_vpp.tests.protocols.FrameTestCase* (page 10)

setUp()
Hook method for setting up the test fixture before exercising it.

test_display_destination()

test_display_ethertype()

test_display_source()

test_ethernet_length()

test_l3_payload_type()

test_unknown_payload_type()

test_validate_destination()

test_validate_ethertype()

test_validate_payload()

test_validate_source()

dhcpkit_vpp.tests.protocols.test_layer3 module

Test whether layer 3 parsing and generating works

class dhcpkit_vpp.tests.protocols.test_layer3.**IPv6TestCase**(*methodName='runTest'*)
Bases: *dhcpkit_vpp.tests.protocols.FrameTestCase* (page 10)

setUp()
Hook method for setting up the test fixture before exercising it.

test_ipv6_length()

test_l4_payload_type()

test_protocol_version()

test_trailing_data()

test_unknown_payload_type()

test_validate_destination()

test_validate_flow_label()

test_validate_hop_limit()

test_validate_next_header()

test_validate_payload()

test_validate_source()

test_validate_traffic_class()

class dhcpkit_vpp.tests.protocols.test_layer3.**UnknownLayer3PacketTestCase**(*methodName='runT*
Bases: *dhcpkit_vpp.tests.protocols.FrameTestCase* (page 10)

setUp()
Hook method for setting up the test fixture before exercising it.

test_pseudo_header()

dhcpkit_vpp.tests.protocols.test_layer4 module

Test whether layer 4 parsing and generating works

class dhcpkit_vpp.tests.protocols.test_layer4.**UDPTestCase**(*methodName='runTest'*)
Bases: *dhcpkit_vpp.tests.protocols.FrameTestCase* (page 10)

setUp()
Hook method for setting up the test fixture before exercising it.

test_checksum_calculation()

test_save_with_checksum_calculation()

test_save_zero_checksum()

test_udp_length()

test_validate_checksum()

test_validate_destination_port()

test_validate_payload()

test_validate_source_port()

class dhcpkit_vpp.tests.protocols.test_layer4.**UnknownLayer4ProtocolTestCase**(*methodName='ru*
Bases: *dhcpkit_vpp.tests.protocols.FrameTestCase* (page 10)

setUp()
Hook method for setting up the test fixture before exercising it.

test_length()

1.2.2 Submodules

dhcpkit_vpp.vpp_papi module

1.3 Changes per version

1.3.1 1.0.1 - 2017-06-21

Fixes

- Package missing component.xml

1.3.2 1.0.0 - 2017-06-21

New features

- Initial release

1.4 Applicable copyright licences

1.4.1 DHCPKit License

1.4.2 VPP-PAPI module license

Python Module Index

d

Index

O

P

S

T

U

V