IDR Working Group
Internet-Draft
Updates: I-D.ietf-idr-rfc5575bis (if approved)
Intended status: Standards Track
Expires: May 6, 2021

C. Loibl, Ed.
next layer Telekom GmbH
R. Raszuk, Ed.
Bloomberg LP
S. Hares, Ed.
Huawei

November 2, 2020

Dissemination of Flow Specification Rules for IPv6
draft-ietf-idr-flow-spec-v6-19

Abstract

Dissemination of Flow Specification Rules I-D.ietf-idr-rfc5575bis provides a Border Gateway Protocol extension for the propagation of traffic flow information for the purpose of rate limiting or filtering IPv4 protocol data packets.

This document extends I-D.ietf-idr-rfc5575bis with IPv6 functionality. It also updates I-D.ietf-idr-rfc5575bis by changing the IANA Flow Spec Component Types registry.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
1.1. Definitions of Terms Used in This Memo 3
2. IPv6 Flow Specification encoding in BGP 3
3. IPv6 Flow Specification components 3
3.1. Type 1 - Destination IPv6 Prefix 4
3.2. Type 2 - Source IPv6 Prefix 4
3.3. Type 3 - Upper-Layer Protocol 4
3.4. Type 7 - ICMPv6 Type 5
3.5. Type 8 - ICMPv6 Code 5
3.6. Type 12 - Fragment 6
3.7. Type 13 - Flow Label (new) 6
3.8. Encoding Example 7
4. Ordering of Flow Specifications 8
5. Validation Procedure 9
6. IPv6 Traffic Filtering Action changes 9
6.1. Redirect IPv6 (rt-redirect-ipv6) Type/Sub-Type 0x80/TBD . 9
7. Security Considerations 9
8. IANA Considerations 10
8.1. Flow Spec IPv6 Component Types 10
8.1.1. Registry Template 10
8.1.2. Registry Contents 10
8.2. Extended Community Flow Spec IPv6 Actions 12
9. Acknowledgements 13
10. Contributors 13
11. References 13
11.1. Normative References 13
11.2. URIs 14
Appendix A. Example python code: flow_rule_cmp_v6 14
Authors' Addresses 18

1. Introduction

The growing amount of IPv6 traffic in private and public networks requires the extension of tools used in IPv4-only networks to be also capable of supporting IPv6 data packets.

This document analyzes the differences of IPv6 [RFC8200] flows description from those of traditional IPv4 packets and propose a subset of new Border Gateway Protocol [RFC4271] encoding formats to

enable Dissemination of Flow Specification Rules [I-D.ietf-idr-rfc5575bis] for IPv6.

This specification is an extension of the base [I-D.ietf-idr-rfc5575bis]. It only defines the delta changes required to support IPv6 while all other definitions and operation mechanisms of Dissemination of Flow Specification Rules will remain in the main specification and will not be repeated here.

1.1. Definitions of Terms Used in This Memo

AFI - Address Family Identifier.

AS - Autonomous System.

NLRI - Network Layer Reachability Information.

SAFI - Subsequent Address Family Identifier.

VRF - Virtual Routing and Forwarding instance.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. IPv6 Flow Specification encoding in BGP

[I-D.ietf-idr-rfc5575bis] defines SAFIs 133 (Dissemination of Flow Specification) and 134 (L3VPN Dissemination of Flow Specification) in order to carry the corresponding Flow Specification.

Implementations wishing to exchange IPv6 Flow Specifications MUST use BGP's Capability Advertisement facility to exchange the Multiprotocol Extension Capability Code (Code 1) as defined in [RFC4760]. The (AFI, SAFI) pair carried in the Multiprotocol Extension Capability MUST be: (AFI=2, SAFI=133) for IPv6 Flow Specification, and (AFI=2, SAFI=134) for VPNv6 Flow Specification.

3. IPv6 Flow Specification components

The encoding of each of the components begins with a type field (1 octet) followed by a variable length parameter. The following sections define component types and parameter encodings for IPv6.

Types 4, 5, 6, 9, 10 and 11, as defined in [I-D.ietf-idr-rfc5575bis], also apply to IPv6. Note that IANA is requested to update the "Flow

Spec Component Types" registry in order to contain both IPv4 and IPv6 Flow Specification component type numbers in a single registry (Section 8).

3.1. Type 1 - Destination IPv6 Prefix

Encoding: <type (1 octet), length (1 octet), offset (1 octet), pattern (variable), padding(variable) >

Defines the destination prefix to match. The offset has been defined to allow for flexible matching to portions of an IPv6 address where one is required to skip over the first N bits of the address (these bits skipped are often indicated as "don't care" bits). This can be especially useful where part of the IPv6 address consists of an embedded IPv4 address and matching needs to happen only on the embedded IPv4 address. The encoded pattern contains enough octets for the bits used in matching (length minus offset bits).

length - The length field indicates the N-th most significant bit in the address where bitwise pattern matching stops.

offset - The offset field indicates the number of most significant address bits to skip before bitwise pattern matching starts.

pattern - Contains the matching pattern. The length of the pattern is defined by the number of bits needed for pattern matching (length minus offset).

padding - The minimum number of bits required to pad the component to an octet boundary. Padding bits MUST be 0 on encoding and MUST be ignored on decoding.

Length = Offset = 0 matches every address, otherwise Length MUST be in the range Offset < Length < 129 or the component is malformed.

3.2. Type 2 - Source IPv6 Prefix

Encoding: <type (1 octet), length (1 octet), offset (1 octet), pattern (variable), padding(variable) >

Defines the source prefix to match. The length, offset, pattern and padding are the same as in Section 3.1

3.3. Type 3 - Upper-Layer Protocol

Encoding: <type (1 octet), [numeric_op, value]+>

Contains a list of {numeric_op, value} pairs that are used to match the first Next Header value octet in IPv6 packets that is not an extension header and thus indicates that the next item in the packet is the corresponding upper-layer header (see [RFC8200] Section 4).

This component uses the Numeric Operator (numeric_op) described in [I-D.ietf-idr-rfc5575bis] Section 4.2.1.1. Type 3 component values SHOULD be encoded as single octet (numeric_op len=00).

Note: While IPv6 allows for more than one Next Header field in the packet, the main goal of the Type 3 Flow Specification component is to match on the first upper-layer IP protocol value. Therefore the definition is limited to match only on this specific Next Header field in the packet.

3.4. Type 7 - ICMPv6 Type

Encoding: <type (1 octet), [numeric_op, value]+>

Defines a list of {numeric_op, value} pairs used to match the type field of an ICMPv6 packet (see also [RFC4443] Section 2.1).

This component uses the Numeric Operator (numeric_op) described in [I-D.ietf-idr-rfc5575bis] Section 4.2.1.1. Type 7 component values SHOULD be encoded as single octet (numeric_op len=00).

In case of the presence of the ICMPv6 Type component only ICMPv6 packets can match the entire Flow Specification. The ICMPv6 Type component, if present, never matches when the packet's upper-layer IP protocol value is not 58 (ICMPv6), if the packet is fragmented and this is not the first fragment, or if the system is unable to locate the transport header. Different implementations may or may not be able to decode the transport header.

3.5. Type 8 - ICMPv6 Code

Encoding: <type (1 octet), [numeric_op, value]+>

Defines a list of {numeric_op, value} pairs used to match the code field of an ICMPv6 packet (see also [RFC4443] Section 2.1).

This component uses the Numeric Operator (numeric_op) described in [I-D.ietf-idr-rfc5575bis] Section 4.2.1.1. Type 8 component values SHOULD be encoded as single octet (numeric_op len=00).

In case of the presence of the ICMPv6 Code component only ICMPv6 packets can match the entire Flow Specification. The ICMPv6 code component, if present, never matches when the packet's upper-layer IP

protocol value is not 58 (ICMPv6), if the packet is fragmented and this is not the first fragment, or if the system is unable to locate the transport header. Different implementations may or may not be able to decode the transport header.

3.6. Type 12 - Fragment

Encoding: <type (1 octet), [bitmask_op, bitmask]+>

Defines a list of {bitmask_op, bitmask} pairs used to match specific IP fragments.

This component uses the Bitmask Operator (bitmask_op) described in [I-D.ietf-idr-rfc5575bis] Section 4.2.1.2. The Type 12 component bitmask MUST be encoded as single octet bitmask (bitmask_op len=00).

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
| 0 | 0 | 0 | 0 |LF |FF |IsF| 0 |
+---+---+---+---+---+---+---+---+
```

Figure 1: Fragment Bitmask Operand

Bitmask values:

IsF - Is a fragment other than the first - match if IPv6 Fragment Header ([RFC8200] Section 4.5) Fragment Offset is not 0

FF - First fragment - match if IPv6 Fragment Header ([RFC8200] Section 4.5) Fragment Offset is 0 AND M flag is 1

LF - Last fragment - match if IPv6 Fragment Header ([RFC8200] Section 4.5) Fragment Offset is not 0 AND M flag is 0

0 - MUST be set to 0 on NLRI encoding, and MUST be ignored during decoding

3.7. Type 13 - Flow Label (new)

Encoding: <type (1 octet), [numeric_op, value]+>

Contains a list of {numeric_op, value} pairs that are used to match the 20-bit Flow Label IPv6 header field ([RFC8200] Section 3).

This component uses the Numeric Operator (numeric_op) described in [I-D.ietf-idr-rfc5575bis] Section 4.2.1.1. Type 13 component values SHOULD be encoded as 1-, 2-, or 4-byte quantities (numeric_op len=00, len=01 or len=10).

3.8. Encoding Example

3.8.1. Example 1

The following example demonstrates the prefix encoding for: "packets from ::1234:5678:9A00:0/64-104 to 2001:DB8::/32 and upper-layer-protocol tcp".

```
+--------+----------------------+-------------------------+----------+
| length | destination          | source                  | ul-proto |
+--------+----------------------+-------------------------+----------+
| 0x12   | 01 20 00 20 01 0D B8 | 02 68 40 12 34 56 78 9A | 03 81 06 |
+--------+----------------------+-------------------------+----------+
```

Decoded:

```
+-------+------------+-------------------------------+
| Value |            |                               |
+-------+------------+-------------------------------+
|  0x12 | length     | 18 octets (len<240 1-octet)   |
|  0x01 | type       | Type 1 - Dest. IPv6 Prefix    |
|  0x20 | length     | 32 bit                        |
|  0x00 | offset     | 0 bit                         |
|  0x20 | pattern    |                               |
|  0x01 | pattern    |                               |
|  0x0D | pattern    |                               |
|  0xB8 | pattern    | (no padding needed)           |
|  0x02 | type       | Type 2 - Source IPv6 Prefix   |
|  0x68 | length     | 104 bit                       |
|  0x40 | offset     | 64 bit                        |
|  0x12 | pattern    |                               |
|  0x34 | pattern    |                               |
|  0x56 | pattern    |                               |
|  0x78 | pattern    |                               |
|  0x9A | pattern    | (no padding needed)           |
|  0x03 | type       | Type 3 - upper-layer-proto    |
|  0x81 | numeric_op | end-of-list, value size=1, == |
|  0x06 | value      | 06                            |
+-------+------------+-------------------------------+
```

This constitutes a NLRI with a NLRI length of 18 octets.

Padding is not needed either for the destination prefix pattern (length - offset = 32 bit) or for the source prefix pattern (length - offset = 40 bit), as both patterns end on an octet boundary.

3.8.2. Example 2

The following example demonstrates the prefix encoding for: "all packets from ::1234:5678:9A00:0/65-104 to 2001:DB8::/32".

```
+--------+----------------------+-------------------------+
| length | destination          | source                  |
+--------+----------------------+-------------------------+
| 0x0f   | 01 20 00 20 01 0D B8 | 02 68 41 24 68 ac f1 34 |
+--------+----------------------+-------------------------+
```

Decoded:

```
+-------+-------------+-------------------------------+
| Value |             |                               |
+-------+-------------+-------------------------------+
|  0x0f | length      | 15 octets (len<240 1-octet)   |
|  0x01 | type        | Type 1 - Dest. IPv6 Prefix    |
|  0x20 | length      | 32 bit                        |
|  0x00 | offset      | 0 bit                         |
|  0x20 | pattern     |                               |
|  0x01 | pattern     |                               |
|  0x0D | pattern     |                               |
|  0xB8 | pattern     | (no padding needed)           |
|  0x02 | type        | Type 2 - Source IPv6 Prefix   |
|  0x68 | length      | 104 bit                       |
|  0x41 | offset      | 65 bit                        |
|  0x24 | pattern     |                               |
|  0x68 | pattern     |                               |
|  0xac | pattern     |                               |
|  0xf1 | pattern     |                               |
|  0x34 | pattern/pad | (contains 1 bit padding)      |
+-------+-------------+-------------------------------+
```

This constitutes a NLRI with a NLRI length of 15 octets.

The source prefix pattern is 104 - 65 = 39 bits in length. After the pattern one bit of padding needs to be added so that the component ends on a octet boundary. However, only the first 39 bits are actually used for bitwise pattern matching starting with a 65 bit offset from the topmost bit of the address.

4. Ordering of Flow Specifications

The definition for the order of traffic filtering rules from [I-D.ietf-idr-rfc5575bis] Section 5.1 is reused with new consideration for the IPv6 prefix offset. As long as the offsets are equal, the comparison is the same, retaining longest-prefix-match

semantics. If the offsets are not equal, the lowest offset has precedence, as this flow matches the most significant bit.

The code in Appendix A shows a Python3 implementation of the resulting comparison algorithm. The full code was tested with Python 3.7.2 and can be obtained at https://github.com/stoffi92/draft-ietf-idr-flow-spec-v6/tree/master/flowspec-cmp [1].

5. Validation Procedure

The validation procedure is the same as specified in [I-D.ietf-idr-rfc5575bis] Section 6 with the exception that item a) of the validation procedure should now read as follows:

a) A destination prefix component with offset=0 is embedded in the Flow Specification

6. IPv6 Traffic Filtering Action changes

Traffic Filtering Actions from [I-D.ietf-idr-rfc5575bis] Section 7 can also be applied to IPv6 Flow Specifications. To allow an IPv6 address specific route-target, a new Traffic Filtering Action IPv6 address specific extended community is specified in Section 6.1 below.

6.1. Redirect IPv6 (rt-redirect-ipv6) Type/Sub-Type 0x80/TBD

The redirect IPv6 address specific extended community allows the traffic to be redirected to a VRF routing instance that lists the specified IPv6 address specific route-target in its import policy. If several local instances match this criteria, the choice between them is a local matter (for example, the instance with the lowest Route Distinguisher value can be elected).

This extended community uses the same encoding as the IPv6 address specific Route Target extended community [RFC5701] Section 2 with the high-order octet of the Type always set to 0x80 and the Sub-Type always TBD.

Interferes with: All BGP Flow Specification redirect Traffic Filtering Actions (with itself and those specified in [I-D.ietf-idr-rfc5575bis] Section 7.4).

7. Security Considerations

This document extends the functionality in [I-D.ietf-idr-rfc5575bis] to be applicable to IPv6 data packets. The same Security Considerations from [I-D.ietf-idr-rfc5575bis] now also apply to IPv6

networks. Otherwise, no new security issues are added to the BGP protocol.

8. IANA Considerations

This section complies with [RFC7153].

8.1. Flow Spec IPv6 Component Types

IANA has created and maintains a registry entitled "Flow Spec Component Types". IANA is requested to add [this document] to the reference for this registry. Furthermore the registry should be rewritten to also contain the IPv6 Flow Specification Component Types as described below. The registration procedure should remain unchanged.

8.1.1. Registry Template

Type Value:
Contains the assigned Flow Specification component type value.

IPv4 Name:
Contains the associated IPv4 Flow Specification component name as specified in [I-D.ietf-idr-rfc5575bis].

IPv6 Name:
Contains the associated IPv6 Flow Specification component name as specified in this document.

Reference:
Contains referenced to the specifications.

8.1.2. Registry Contents

+ Type Value: 0
+ IPv4 Name: Reserved
+ IPv6 Name: Reserved
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 1
+ IPv4 Name: Destination Prefix
+ IPv6 Name: Destination IPv6 Prefix
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 2
+ IPv4 Name: Source Prefix

+ IPv6 Name: Source IPv6 Prefix
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 3
+ IPv4 Name: IP Protocol
+ IPv6 Name: Upper-Layer Protocol
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 4
+ IPv4 Name: Port
+ IPv6 Name: Port
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 5
+ IPv4 Name: Destination Port
+ IPv6 Name: Destination Port
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 6
+ IPv4 Name: Source Port
+ IPv6 Name: Source Port
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 7
+ IPv4 Name: ICMP Type
+ IPv6 Name: ICMPv6 Type
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 8
+ IPv4 Name: ICMP Code
+ IPv6 Name: ICMPv6 Code
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 9
+ IPv4 Name: TCP flags
+ IPv6 Name: TCP flags
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 10
+ IPv4 Name: Packet length

+ IPv6 Name: Packet length
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 11
+ IPv4 Name: DSCP
+ IPv6 Name: DSCP
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 12
+ IPv4 Name: Fragment
+ IPv6 Name: Fragment
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

+ Type Value: 13
+ IPv4 Name: Unassigned
+ IPv6 Name: Flow Label
+ Reference: [this document]

+ Type Value: 14-254
+ IPv4 Name: Unassigned
+ IPv6 Name: Unassigned
+ Reference:

+ Type Value: 255
+ IPv4 Name: Reserved
+ IPv6 Name: Reserved
+ Reference: [I-D.ietf-idr-rfc5575bis] [this document]

8.2. Extended Community Flow Spec IPv6 Actions

IANA maintains a registry entitled "Generic Transitive Experimental Use Extended Community Sub-Types". For the purpose of this work, IANA is requested to assign a new value:

Sub-Type Value	Name	Reference
TBD	Flow spec rt-redirect-ipv6 format	[this document]

Table 1: Registry: Generic Transitive Experimental Use Extended Community Sub-Types

9. Acknowledgements

Authors would like to thank Pedro Marques, Hannes Gredler, Bruno Rijsman, Brian Carpenter, and Thomas Mangin for their valuable input.

10. Contributors

Danny McPherson
Verisign, Inc.

Email: dmcpherson@verisign.com

Burjiz Pithawala
Individual

Email: burjizp@gmail.com

Andy Karch
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
USA

Email: akarch@cisco.com

11. References

11.1. Normative References

[I-D.ietf-idr-rfc5575bis]
Loibl, C., Hares, S., Raszuk, R., McPherson, D., and M. Bacher, "Dissemination of Flow Specification Rules", draft-ietf-idr-rfc5575bis-26 (work in progress), August 2020.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, DOI 10.17487/RFC4760, January 2007, <https://www.rfc-editor.org/info/rfc4760>.

[RFC5701] Rekhter, Y., "IPv6 Address Specific BGP Extended Community Attribute", RFC 5701, DOI 10.17487/RFC5701, November 2009, <https://www.rfc-editor.org/info/rfc5701>.

[RFC7153] Rosen, E. and Y. Rekhter, "IANA Registries for BGP Extended Communities", RFC 7153, DOI 10.17487/RFC7153, March 2014, <https://www.rfc-editor.org/info/rfc7153>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

11.2. URIs

[1] https://github.com/stoffi92/draft-ietf-idr-flow-spec-v6/tree/master/flowspec-cmp

Appendix A. Example python code: flow_rule_cmp_v6

```
<CODE BEGINS>
"""
Copyright (c) 2020 IETF Trust and the persons identified as authors
```

```
    of draft-ietf-idr-flow-spec-v6. All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, is permitted pursuant to, and subject to the license
    terms contained in, the Simplified BSD License set forth in Section
    4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).
    """

    import itertools
    import collections
    import ipaddress


    EQUAL = 0
    A_HAS_PRECEDENCE = 1
    B_HAS_PRECEDENCE = 2
    IP_DESTINATION = 1
    IP_SOURCE = 2

    FS_component = collections.namedtuple('FS_component',
                                          'component_type value')


    class FS_IPv6_prefix_component:
        def __init__(self, prefix, offset=0,
                     component_type=IP_DESTINATION):
            self.offset = offset
            self.component_type = component_type
            # make sure if offset != 0 that non of the
            # first offset bits are set in the prefix
            self.value = prefix
            if offset != 0:
                i = ipaddress.IPv6Interface(
                    (self.value.network_address, offset))
                if i.network.network_address != \
                    ipaddress.ip_address('0::0'):
                    raise ValueError('Bits set in the offset')


    class FS_nlri(object):
        """
        FS_nlri class implementation that allows sorting.

        By calling .sort() on a array of FS_nlri objects these
        will be sorted according to the flow_rule_cmp algorithm.

        Example:
```

```
        nlri = [ FS_nlri(components=[
                 FS_component(component_type=4,
                              value=bytearray([0,1,2,3,4,5,6])),
                 ]),
                 FS_nlri(components=[
                 FS_component(component_type=5,
                              value=bytearray([0,1,2,3,4,5,6])),
                 FS_component(component_type=6,
                              value=bytearray([0,1,2,3,4,5,6])),
                 ]),
                ]
        nlri.sort() # sorts the array accorinding to the algorithm
        """
        def __init__(self, components = None):
            """
            components: list of type FS_component
            """
            self.components = components

        def __lt__(self, other):
            # use the below algorithm for sorting
            result = flow_rule_cmp_v6(self, other)
            if result == B_HAS_PRECEDENCE:
                return True
            else:
                return False


    def flow_rule_cmp_v6(a, b):
        """
        Implementation of the flowspec sorting algorithm in
        draft-ietf-idr-flow-spec-v6.
        """
        for comp_a, comp_b in itertools.zip_longest(a.components,
                                                    b.components):
            # If a component type does not exist in one rule
            # this rule has lower precedence
            if not comp_a:
                return B_HAS_PRECEDENCE
            if not comp_b:
                return A_HAS_PRECEDENCE
            # Higher precedence for lower component type
            if comp_a.component_type < comp_b.component_type:
                return A_HAS_PRECEDENCE
            if comp_a.component_type > comp_b.component_type:
                return B_HAS_PRECEDENCE
            # component types are equal -> type specific comparison
            if comp_a.component_type in (IP_DESTINATION, IP_SOURCE):
```

```
                if comp_a.offset < comp_b.offset:
                    return A_HAS_PRECEDENCE
                if comp_a.offset < comp_b.offset:
                    return B_HAS_PRECEDENCE
                # both components have the same offset
                # assuming comp_a.value, comp_b.value of type
                # ipaddress.IPv6Network
                # and the offset bits are reset to 0 (since they are
                # not represented in the NLRI)
                if comp_a.value.overlaps(comp_b.value):
                    # longest prefixlen has precedence
                    if comp_a.value.prefixlen > \
                        comp_b.value.prefixlen:
                        return A_HAS_PRECEDENCE
                    if comp_a.value.prefixlen < \
                        comp_b.value.prefixlen:
                        return B_HAS_PRECEDENCE
                    # components equal -> continue with next
                    # component
                elif comp_a.value > comp_b.value:
                    return B_HAS_PRECEDENCE
                elif comp_a.value < comp_b.value:
                    return A_HAS_PRECEDENCE
            else:
                # assuming comp_a.value, comp_b.value of type
                # bytearray
                if len(comp_a.value) == len(comp_b.value):
                    if comp_a.value > comp_b.value:
                        return B_HAS_PRECEDENCE
                    if comp_a.value < comp_b.value:
                        return A_HAS_PRECEDENCE
                    # components equal -> continue with next
                    # component
                else:
                    common = min(len(comp_a.value),
                                 len(comp_b.value))
                    if comp_a.value[:common] > \
                        comp_b.value[:common]:
                        return B_HAS_PRECEDENCE
                    elif comp_a.value[:common] < \
                          comp_b.value[:common]:
                        return A_HAS_PRECEDENCE
                    # the first common bytes match
                    elif len(comp_a.value) > len(comp_b.value):
                        return A_HAS_PRECEDENCE
                    else:
                        return B_HAS_PRECEDENCE
        return EQUAL
```

<CODE ENDS>

Authors' Addresses

Christoph Loibl (editor)
next layer Telekom GmbH
Mariahilfer Guertel 37/7
Vienna 1150
AT

Phone: +43 664 1176414
Email: cl@tix.at

Robert Raszuk (editor)
Bloomberg LP
731 Lexington Ave
New York City, NY 10022
USA

Email: robert@raszuk.net

Susan Hares (editor)
Huawei
7453 Hickory Hill
Saline, MI 48176
USA

Email: shares@ndzh.com